MASTER EN SCIENCES MATHÉMATIQUES

Algorithmes évolutionnaires distribués/coopératifs

BLANCHARD, Julien

Award date:
2015

Awarding institution:
Universite de Namur

UNIVERSITÉ DE NAMUR

Faculté des Sciences

ALGORITHMES ÉVOLUTIONNAIRES DISTRIBUÉS/COOPÉRATIFS

Mémoire présenté pour l'obtention

du grade académique de master en sciences mathématiques à finalité approfondie

Julien BLANCHARD

Juin 2015

UNIVERSITÉ DE NAMUR

Faculté des Sciences

ALGORITHMES ÉVOLUTIONNAIRES DISTRIBUÉS/COOPÉRATIFS

Promoteur :

Annick Sartenaer

Co-promoteur :

Charlotte Beauthier

Mémoire présenté pour l'obtention

du grade académique de master en sciences mathématiques à finalité approfondie

Julien BLANCHARD

Juin 2015

Remerciements

Ce mémoire est le fruit d'un travail de recherche de près de deux ans. J'adresse mes remerciements les plus sincères aux personnes qui m'ont aidé à l'élaboration de celui-ci.

Je tiens d'abord à remercier Annick Sartenaer qui, en tant que promotrice, m'a donné des conseils et des encouragements tout au long de ce mémoire.

Je remercie également Charlotte Beauthier, co-promotrice de ce mémoire, pour sa patience ainsi que l'appui et le soutien qu'elle m'a apportés au cours de mon mémoire et notamment lors de mon stage à Cenaero.

Je remercie également le centre de recherche Cenaero, avec qui ce mémoire est réalisé et au sein duquel j'ai travaillé durant plusieurs mois. Plus particulièrement, je remercie les membres de l'équipe Minamo pour leur accueil chaleureux.

Finalement, j'adresse ce dernier remerciement, mais non le moindre, aux membres de ma famille, pour leur soutien et leurs encouragements précieux tout au long de mes études.

Résumé

Ce mémoire, réalisé en collaboration avec le centre de recherche Cenaero, s'inscrit dans le cadre de l'optimisation de fonctions de grande dimension au moyen d'algorithmes évolutionnaires. De nos jours, ceux-ci sont couramment utilisés dans le monde de l'industrie. Ils permettent d'optimiser de nombreux problèmes tout en ne possédant que peu d'informations sur ceux-ci. Ils perdent toutefois de leur efficacité lorsque la dimension du problème traité devient relativement grande (plus d'une centaine de variables).

L'objectif de ce mémoire est d'investiguer, développer et tester un type avancé d'algorithme évolutionnaire permettant d'optimiser des problèmes de grande dimension. Ces algorithmes, couramment dénommés *algorithmes évolutionnaires distribués* ou *coopératifs*, consistent à diviser le problème original en sous-problèmes de dimensionnalité moindre et ensuite à optimiser chacun des sous-problèmes, ceux-ci coopérant entre eux. Dans le cadre de ce mémoire, nous avons implémenté deux algorithmes dont le fonctionnement s'appuie sur un procédé itératif de décompositions aléatoires. Le second est une variante du premier permettant d'auto-adapter certains paramètres de l'algorithme en cours d'optimisation. Ces implémentations ont été réalisées dans le logiciel Minamo, plate-forme d'optimisation multi-disciplinaire du centre de recherche Cenaero.

Dans le but de tester les performances de ces algorithmes, nous avons élaboré un ensemble de fonctions test multi-dimensionnelles présentant des caractéristiques variées et couvrant un large éventail de problèmes d'optimisation mono-objectif sans contraintes. Les algorithmes distribués ont ensuite été appliqués à chacune des fonctions test de dimension 50 et 500 et leur efficacité pour des problèmes de grande dimension a ainsi pu être mise en évidence.

Abstract

This master's thesis, carried out in collaboration with the Cenaero research center, is devoted to the optimization of large scale problems with evolutionary algorithms. Nowadays, these algorithms are commonly used in the world of industry. They allow to optimize many problems while having few information on them. However, they loose their efficiency once the dimensionality of the problem at hand becomes relatively large (more than one hundred variables).

The purpose of this master's thesis is to investigate, develop and validate an advanced kind of algorithm which allows to optimize large scale problems. These algorithms, commonly called *cooperative* or *distributed evolutionary algorithms*, consist first in dividing the original problem into subproblems of lower dimensionality and then in optimizing each of the subproblems, each of them cooperating with the other ones. In the framework of this master's thesis, two algorithms based on an iterative process of random decompositions have been implemented. The second one is a variant of the first one which allows to self-adapt some parameters of the algorithm during the optimization. These implementations have been achieved in the Cenaero's multi-disciplinary optimization platform, called MINAMO.

In order to validate and to test the performances of these algorithms, a set of scalable benchmark functions, with a variety of different features covering a wide range of mono-objective unconstrained optimization problems, have been developed. The distributed algorithms were then applied to each of the benchmark functions of size 50 and 500 and their effectiveness on large scale problems has been highlighted.

Table des matières

Introduction

De nos jours, les algorithmes évolutionnaires sont couramment utilisés dans le monde industriel pour résoudre de nombreux problèmes d'optimisation. Les récents progrès réalisés dans ce domaine permettent aujourd'hui d'optimiser des problèmes de plus en plus complexes tout en ne possédant que très peu d'informations sur ceux-ci. Toutefois, ces algorithmes perdent de leur efficacité lorsque la dimension du problème traité devient relativement grande (plus d'une centaine de variables). Ce phénomène, rencontré dans bien des domaines, est communément appelé « Malédiction de la dimension » (Richard Bellman [2]). Une solution afin de faire face à ce phénomène consiste à diviser un problème de grande dimension en sous-problèmes de dimensionnalité moindre. Dans le cadre des algorithmes évolutionnaires, cette approche porte le nom d'*algorithmes évolutionnaires distribués* ou *coopératifs*. Mitchell A. Potter [22] est considéré comme le fondateur de ces algorithmes. Les premières recherches dans ce domaine reposaient sur l'hypothèse, irréaliste, selon laquelle le problème était séparable. Par la suite, diverses voies furent explorées dans le but d'également traiter des problèmes non séparables. L'une d'entre elles, développées par Zhenyu Yang, Ke Tang et Xin Yao [26], s'appuie sur un procédé itératif de décompositions aléatoires.

L'objectif de ce mémoire, mené en collaboration avec le centre de recherche Cenaero, est d'investiguer, développer et tester cette approche permettant d'optimiser des problèmes de grande dimension. Celle-ci repose sur trois points clés : la décomposition, la coopération et l'optimisation. La décomposition du problème en sous-problèmes est sans doute le plus important d'entre eux, la qualité de la solution obtenue en étant hautement dépendante. Quant à la coopération, son principe s'appuie sur l'échange d'informations entre les différents sous-problèmes permettant ainsi une bonne résolution du problème global. Finalement, l'optimisation de chacun des sous-problèmes est réalisée au moyen d'un algorithme évolutionnaire. Deux algorithmes sont implémentés dans le cadre de ce mémoire. Le premier s'appuie sur le procédé décrit par Yang, Tang et Yao [26] tandis que le second, amélioration du premier, a été développé spécifiquement dans le cadre de ce mémoire et permet d'auto-adapter certains paramètres au cours de l'optimisation. L'implémentation de ces algorithmes évolutionnaires distribués est réalisée au sein du logiciel Minamo, plate-forme d'optimisation multi-disciplinaire du centre de recherche Cenaero. Ils font usage de nombreuses fonctionnalités implémentées dans le logiciel, dont notamment l'algorithme évolutionnaire sur lequel repose le fonctionnement du logiciel. Le bon fonctionnement de ces algorithmes sur des problèmes de

grande dimension (50 et 500 D) est également mis en évidence au moyen de nombreuses fonctions test multi-dimensionnelles présentant des caractéristiques variées et couvrant un large éventail de problèmes d'optimisation mono-objectif sans contraintes.

Ce mémoire se présente en sept chapitres. Dans le Chapitre 1, nous introduisons quelques notions de base d'optimisation ainsi que le fonctionnement d'un type particulier d'algorithme évolutionnaire, à savoir l'algorithme génétique. Le Chapitre 2 présente le centre de recherche Cenaero avec lequel nous avons collaboré pour ce mémoire ainsi que le logiciel MINAMO, code au sein duquel sont implémentés les algorithmes développés dans ce mémoire. Le Chapitre 3 reprend un état de l'art des algorithmes évolutionnaires distribués. Nous y expliquons notamment le fonctionnement des étapes de décomposition, de coopération et d'optimisation. Dans le Chapitre 4, nous décrivons l'ensemble des problèmes test utilisés pour valider l'implémentation de nos algorithmes évolutionnaires distribués. Il s'agit de problèmes mono-objectif sans contraintes. Nous y présentons également les outils à l'aide desquels les résultats sont analysés. Les Chapitres 5 et 6 proposent une description des algorithmes implémentés dans le cadre de ce mémoire, à savoir l'algorithme `Minamo DistrEA` pour *Minamo Distributed Evolutionary Algorithm* et l'algorithme `Minamo SADistrEA` pour *Minamo Self-Adaptive Distributed Evolutionary Algorithm*. Nous testons ensuite les performances de ceux-ci sur l'ensemble des fonctions test. Dans le Chapitre 7, nous nous intéressons aux problèmes d'optimisation avec contraintes. Dans un premier temps, nous présentons les différents procédés permettant de résoudre des problèmes contraints à l'aide d'algorithmes évolutionnaires. Par la suite, nous abordons la gestion des contraintes par le logiciel MINAMO et nous testons l'algorithme `Minamo SADistrEA` sur quelques problèmes contraints. Finalement, nous terminons par quelques conclusions et perspectives.

Chapitre 1

Notions d'optimisation

Ce premier chapitre a pour but de poser le cadre de travail de ce mémoire. L'objectif de celui-ci est d'étudier et d'implémenter des algorithmes d'optimisation distribués. Les concepts liés à ce type d'algorithmes sont décrits au Chapitre 3. Au préalable, nous abordons, dans ce chapitre, l'optimisation de manière générale. Nous commençons par définir la notion de problème d'optimisation. Nous en explicitons ensuite différents types ainsi que différents types d'algorithmes. Enfin, nous détaillons le fonctionnement d'une variété particulière d'algorithme évolutionnaire, à savoir les algorithmes génétiques car c'est au moyen de ces derniers que seront étudiés et implémentés les algorithmes d'optimisation distribués.

1.1 Les problèmes d'optimisation

Dans cette section, nous présentons, sur base des ouvrages [3] et [19], la notion de problème d'optimisation.

Nous sommes tous, consciemment ou non, confrontés au quotidien à des problèmes d'optimisation. Les entreprises qui produisent des biens essaient notamment de maximiser leur profit en fonction des coûts de production ainsi que de la demande. Un livreur devant fournir plusieurs magasins tente de minimiser la distance qu'il parcourt sur la route tout en passant par chacun des magasins. Dans les sciences, l'optimisation joue aussi un rôle important. En physique, les systèmes tendent vers un état d'énergie minimale. L'optimisation intervient aussi dans d'autres domaines des mathématiques. En statistiques par exemple, la résolution du problème de maximum de vraisemblance n'est autre qu'un problème d'optimisation. Bref, l'optimisation est une notion qui nous entoure mais concrètement, qu'est-ce que l'optimisation ? Pour répondre à cette question, nous nous basons sur la notion d'*optimum* donnée par Larousse (2000).

> *« Optimum (du latin optimus, le meilleur) : état, degré de développement de quelque chose jugé le plus favorable au regard des circonstances données. »*

Pour lier ces notions aux mathématiques, il est nécessaire d'opérer une étape de modélisation : tout d'abord, en identifiant les *variables de décision*, c'est-à-dire les variables

qui décrivent le système que l'on souhaite optimiser, ensuite, en définissant une *fonction objectif* f, c'est-à-dire une fonction qui décrit quantitativement l'état d'un système. Afin que l'état de ce système soit le plus favorable, on choisit, selon les cas, de maximiser ou de minimiser cette fonction. Enfin, les circonstances sont décrites au moyen de *contraintes*. Ce sont des égalités et/ou des inégalités que les variables de décision doivent satisfaire. Les valeurs attribuées aux variables vérifiant ces contraintes sont dites *admissibles*. La description mathématique d'un problème d'optimisation est alors écrite sous la forme suivante :

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.c. } \begin{cases} g_i(x) \leq 0 & \forall i = 1, 2, \ldots, m \\ h_j(x) = 0 & \forall j = 1, 2, \ldots, \tilde{m}. \end{cases}$$

Il s'agit ici d'un problème de minimisation pour lequel on a n variables de décision réelles ainsi que m *contraintes d'inégalités* g_i $(i = 1, 2, \ldots, m)$ et $\tilde{m}$ *contraintes d'égalités* h_j $(j = 1, 2, \ldots, \tilde{m})$.

1.2 Les types de problèmes d'optimisation

Selon le type de problème et selon les informations dont on dispose sur celui-ci, différentes méthodes peuvent être utilisées pour le résoudre. Dans cette section, nous explicitons, sur base de l'ouvrage [19], quelques critères qui permettent de classer les problèmes d'optimisation.

Une première distinction peut s'opérer au niveau des variables de décision. En effet, il n'est pas toujours possible de travailler avec des variables prenant des valeurs dans un intervalle continu. Si, par exemple, une variable de décision exprime le nombre d'employés dans une entreprise, celle-ci doit nécessairement prendre des valeurs entières. On fait dès lors la distinction entre l'*optimisation continue* et l'*optimisation discrète*. Bien que la tentation de résoudre un problème d'optimisation discrète par une méthode pour des problèmes continus et en arrondissant ensuite le résultat à l'entier le plus proche soit grande, cela ne garantit pas que la solution obtenue soit proche de la solution optimale. Il s'agit donc bien de deux types de problèmes d'optimisation distincts qui possèdent chacun leurs propres techniques de résolution.

En ce qui concerne l'optimisation continue, il est également important de distinguer l'*optimisation locale* de l'*optimisation globale*. Un *minimum/maximum local* est un *point*, c'est-à-dire un ensemble de valeurs attribuées aux variables de décision, auquel la valeur de la fonction objectif est plus petite/grande qu'à tous les autres points admissibles dans un voisinage tandis qu'un *minimum/maximum global* est un point auquel la valeur de la fonction objectif est plus petite/grande qu'à l'ensemble de tous les autres points admissibles. Une illustration de ces notions est fournie à la FIGURE 1.1. Il est généralement plus simple de trouver un optimum local d'une fonction mais ce n'est pas suffisant pour certaines applications où l'optimum global est nécessaire. Celui-ci est plus difficile à déterminer. Dans le cas de problèmes convexes (pour lesquels la fonction objectif et les contraintes d'inégalité sont convexes tandis que les contraintes d'égalité

sont linéaires), il y a un seul optimum local. Celui-ci coïncide donc avec l'optimum global qui peut dès lors être facilement déterminé.

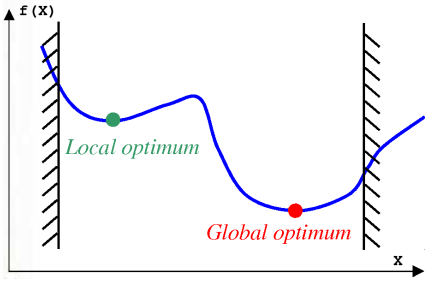


FIGURE 1.1 – Illustration des notions de minimum local et global - Source : [4]

Une autre différence concerne les contraintes du problème. Certains possèdent des contraintes tandis que d'autres pas. Les contraintes expriment les relations entre les variables de décision qui doivent être satisfaites. Il s'agit généralement d'inégalités et d'égalités. Dans certains cas, on préfère transformer un problème avec contraintes en un problème sans contraintes. Pour ce faire, on ajoute à la fonction objectif à minimiser une fonction de pénalité qui est nulle aux points admissibles et strictement positive aux points non admissibles. Ces fonctions de pénalité sont multipliées par des constantes positives qui permettent de donner un poids à la violation des différentes contraintes par rapport à la fonction objectif.

Bien sûr, il existe de nombreux autres types de problèmes d'optimisation. Des informations complémentaires peuvent notamment être trouvées sur le site internet [17].

1.3 Les algorithmes d'optimisation

Au niveau des algorithmes d'optimisation, nous pouvons notamment les distinguer sur base de leur caractère local ou global ou encore sur base de leur caractère déterministe ou stochastique.

Les algorithmes dits *locaux* fournissent un minimum local d'un problème d'optimisation. Parmi les plus connus, on retrouve les algorithmes basés sur une méthode de recherche linéaire dont le fonctionnement est décrit à l'Algorithme 1.1 (source [19]). Selon le choix opéré pour déterminer la direction de descente et la longueur du pas, on obtient différentes méthodes. Nous pouvons citer, entre autres, la méthode de *la plus forte pente* et la méthode de *Newton*. Quant aux algorithmes dits *globaux*, ils permettent de déterminer le minimum global d'un problème d'optimisation. Parmi ceux-ci, on retrouve l'algorithme du *recuit simulé* qui est inspiré du processus du même nom utilisé en métallurgie.

Algorithme 1.1 : Méthode de recherche linéaire

Choisir un point initial x_0 et poser $k = 0$;
tant que $||\nabla f(x_k)|| > \epsilon$ **faire**
  Déterminer une direction de descente d_k (i.e., tq. $\nabla f(x_k)^T d_k < 0$) ;
  Déterminer une longueur de pas α_k ;
  Mettre à jour : $x_{k+1} = x_k + \alpha_k d_k$;
  Incrémenter : $k = k + 1$;
fin

Certains algorithmes génèrent et utilisent des variables aléatoires. Deux exécutions successives de tels algorithmes peuvent donc donner des résultats différents. Ce sont les algorithmes *stochastiques*, parmi lesquels on retrouve les algorithmes d'optimisation par *essaim de particules*, de *recherche tabou*, ainsi que les *algorithmes génétiques*. Au contraire, pour les processus *déterministes*, tout est déterminé au préalable. Deux exécutions successives de ces algorithmes donnent donc des résultats identiques. Parmi ceux couramment utilisés, on peut citer l'algorithme du *simplexe*, du *gradient conjugué* et l'algorithme par *séparation et évaluation*.

1.4 Les algorithmes génétiques

Dans ce mémoire, nous nous intéressons à la résolution de problèmes d'optimisation au moyen d'algorithmes évolutionnaires et plus particulièrement au moyen d'algorithmes génétiques. Ceux-ci constituent une sous-classe d'algorithmes évolutionnaires. Nous commençons par situer ce type d'algorithmes dans le schéma présenté aux sections précédentes. Les algorithmes génétiques traitent de problèmes d'optimisation discrets ou continus avec ou sans contraintes et recherchent un optimum global à l'aide d'un processus stochastique. Ils présentent aussi l'avantage non négligeable de ne nécessiter que très peu d'informations sur la fonction à optimiser. En effet, seule l'évaluation de la fonction objectif en certains points est nécessaire. Contrairement à beaucoup d'autres algorithmes, nous n'aurons donc pas besoin de la connaissance des dérivées premières et/ou secondes pour pouvoir les appliquer. Nous entamons cette section en abordant l'origine des algorithmes génétiques et leurs liens avec la biologie. Nous en expliquons ensuite le principe de base. Enfin, nous analysons plus en détail les différentes étapes de ces algorithmes. Dans cette section, nous nous sommes basés sur les travaux de [1], [11], [13] et [15].

1.4.1 Origines et liens avec la biologie

L'idée principale des algorithmes génétiques se base sur la théorie de l'évolution de Charles Darwin. Selon cette théorie, les espèces présentes aujourd'hui sur terre sont le fruit d'une longue évolution qui s'est opérée par le biais de la reproduction et de la mutation des individus des différentes espèces ainsi que d'un processus de sélection

naturelle. Ce sont les espèces les mieux adaptées à leur environnement qui se perpétuent tandis que les autres espèces disparaissent.

Celui que l'on considère comme le fondateur des algorithmes génétiques est J. H. Holland [12], informaticien et professeur à l'Université du Michigan dans les années 60. Il eut l'idée d'appliquer des opérateurs de reproduction, de sélection et de mutation à une population d'individus. Son cadre de travail restait dans la biologie. En effet, son but était d'analyser des processus auto-adaptatifs au sein de populations, animales entre autres. Un de ses étudiants, David Goldberg, eut l'idée d'utiliser les notions développées par Holland dans le but de résoudre des problèmes mathématiques. Les algorithmes génétiques que nous utilisons en optimisation de nos jours sont le fruit de la voie ouverte par ces deux américains.

Les algorithmes génétiques étant basés sur des notions de biologie, nous allons, afin de les expliquer au mieux, procéder à une explication quelque peu simplifiée de différents concepts biologiques. Ceux-ci sont repris ci-dessous :

- Un *chromosome* est une séquence d'ADN. Ce sont dans des séquences d'ADN que sont contenues les informations génétiques. Pour un être humain, ces informations peuvent par exemple être la couleur des cheveux, des yeux, le teint de la peau, etc.
- Chaque chromosome est composé de différents *gènes*. Chaque gène code une caractéristique, par exemple la couleur des yeux.
- Un gène peut prendre différentes valeurs, par exemple brun, bleu ou vert. Ces valeurs sont appelées *allèles*.
- L'être humain est un individu *diploïde*, ce qui signifie que ses chromosomes sont toujours groupés par paires, un chromosome hérité de la mère est groupé avec un chromosome hérité du père. Ainsi, chaque caractéristique est codée par deux gènes. Par exemple, pour la couleur des yeux, si les deux allèles sont identiques, la couleur des yeux de l'individu est celle correspondant aux allèles. S'ils sont différents, la couleur des yeux est celle correspondant à l'allèle dit *dominant*. L'autre allèle est appelé *récessif*.
- La caractéristique observée est appelée *phénotype* tandis que l'information contenue dans les gènes correspondant à cette caractéristique est appelée *génotype*.
- La *viabilité* est la capacité ou non d'un individu à s'adapter à son environnement.

Sur base de ces quelques notions, nous pouvons expliquer le mécanisme de l'évolution. Nous allons nous intéresser à l'être humain. Celui-ci possède 23 paires de chromosomes. Lors de la reproduction entre deux individus, les 23 chromosomes d'un individu se groupent par paires avec les 23 chromosomes de l'autre. De plus, l'allèle d'un gène peut être modifié suite à une mutation. Les mécanismes de reproduction et de mutation produisent ainsi un enfant dont les génotypes et/ou les phénotypes sont différents de ceux des parents. Le processus de la sélection naturelle se traduit au travers de la viabilité. Seuls les individus les mieux adaptés à leur environnement sont capables de se perpétuer, les autres disparaissent.

1.4.2 Principe de base

Afin d'expliquer le fonctionnement des algorithmes génétiques, nous commençons par faire une analogie entre la biologie et notre cadre de travail.

- Une population est un ensemble (fini) d'individus, un individu étant une solution potentielle du problème d'optimisation. Quelques simplifications sont effectuées par rapport à la biologie. En effet, nous supposons que les individus sont *haploïdes*, ce qui signifie que chaque caractéristique est codée par un seul gène et non plus deux comme pour l'être humain. Nous supposons également que chaque individu est caractérisé par un seul chromosome. Chaque solution potentielle est donc représentée par un chromosome.
- Chaque chromosome ne contient pas exactement une solution potentielle mais un codage de celle-ci. Un codage fréquemment utilisé est le codage binaire. C'est sur ce codage que s'appliqueront les opérateurs génétiques de reproduction et de mutation. Il existe d'autres alternatives au codage binaire mais nous effectuerons la suite de nos explications sur base de celui-ci.
- Tout comme un chromosome est composé de gènes, le codage d'une solution potentielle, que nous appellerons individu afin de simplifier les choses, est composé de bits.
- Chaque bit peut prendre deux valeurs, 0 ou 1. D'un point de vue biologique, ces valeurs correspondent aux allèles.
- Le phénotype est la représentation en base 10 d'une solution, tandis que son génotype est sa représentation en binaire.
- La viabilité est exprimée par le *fitness* d'un individu, c'est-à-dire la valeur de la fonction objectif pour cet individu.

Les concepts étant bien définis, nous pouvons à présent décrire le fonctionnement des algorithmes génétiques.

Il faut d'abord générer aléatoirement une population initiale d'individus et évaluer le fitness de chacun de ceux-ci. Sur base des évaluations du fitness, une sélection est ensuite opérée afin de choisir les meilleurs individus. De nouveaux individus sont alors obtenus en appliquant des opérateurs génétiques de reproduction et de mutation aux meilleurs individus. Par après, il faut générer la population suivante. Pour ce faire, une sélection est opérée parmi la population actuelle et les individus produits par les étapes de reproduction et de mutation. Le même processus est alors réitéré sur base de cette nouvelle population jusqu'à atteindre un critère d'arrêt qui est généralement un nombre d'itérations fixé au préalable. Ces itérations sont aussi appelées *générations*. Une illustration du fonctionnement de cet algorithme est proposé à la Figure 1.2.

Notons qu'au travers des différentes étapes de cet algorithme, on s'attend à produire de meilleurs individus lors de chaque génération. Au bout d'un certain nombre d'itérations, tous les individus sont donc très proches de l'optimum recherché. Différents processus sont possibles lors de chaque étape de l'algorithme, les plus courants sont présentés dans les sections suivantes.

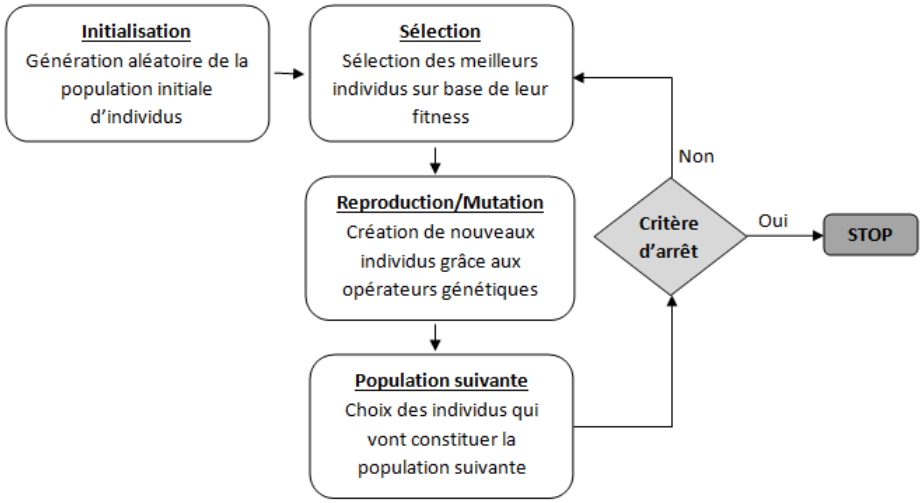


FIGURE 1.2 – Schéma de l'algorithme génétique

1.4.3 Sélection

L'étape de sélection consiste à choisir les individus qui vont participer au processus de reproduction. On choisit les meilleurs individus sur base de leur fitness afin d'obtenir, au fil des générations, des individus avec un meilleur fitness et ainsi atteindre l'optimum de la fonction objectif. Différentes méthodes existent pour la sélection des individus. La plus connue d'entre elles est la *roulette*. Nous aborderons aussi la méthode de l'*élitisme* ainsi que le *tournoi*.

Pour effectuer la sélection grâce au mécanisme de la roulette, il faut commencer par additionner le fitness de tous les individus et ensuite calculer, pour chacun des individus, le rapport entre son fitness et le fitness total. Ce rapport définit la probabilité qu'un individu soit sélectionné. Remarquons que ce processus de sélection ne s'applique que si la fonction de fitness est à valeur dans les réels positifs. Si ce n'est pas le cas, il suffit de la composer avec une fonction monotone convenablement choisie afin que la nouvelle fonction obtenue soit à valeur dans les réels positifs. Il est possible d'illustrer cette sélection au moyen du jeu de la roulette. En effet, il suffit de considérer une roulette où chacune des cases représente un individu, la largeur de celle-ci étant proportionnelle au fitness de l'individu. Lorsqu'on fait tourner la roulette, l'individu sélectionné est celui pour lequel la bille s'est arrêtée sur sa case. La FIGURE 1.3 illustre ce mécanisme. D'un point de vue algorithmique, la sélection est assez simple. Il suffit de faire correspondre à la probabilité d'un individu d'être choisi, un sous-intervalle de $[0, 1]$. On tire ensuite un nombre aléatoire entre 0 et 1. L'individu choisi est alors l'individu dont le sous-intervalle correspondant contient le nombre aléatoire. Cette méthode est pratique car elle est assez simple mais possède tout de même quelques désavantages. Si, par exemple, un individu a un très grand fitness par rapport au fitness total, il a une très grande probabilité d'être choisi et est donc choisi beaucoup de fois. La population risque donc de converger trop vite vers un seul individu. Le risque est alors de ne pas converger vers

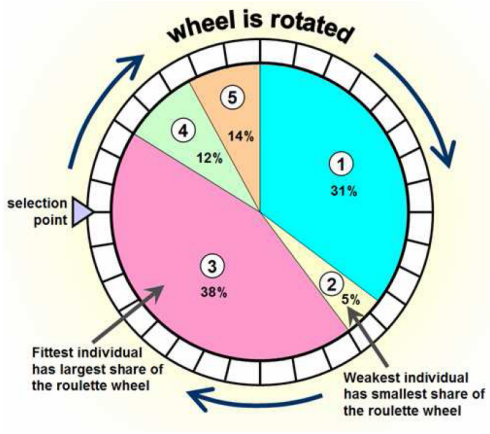


FIGURE 1.3 – Illustration de la sélection par la méthode de la roulette - Source : [18]

l'optimum global du problème mais plutôt vers un optimum local.

Avec le mécanisme de la roulette, il se pourrait que dans certains cas, le ou les meilleurs individus ne soient pas sélectionnés. L'élitisme permet de contourner ce problème. L'élitisme pur consiste à simplement choisir les individus que l'on sélectionne comme étant les individus avec le meilleur fitness. Il est courant de coupler la technique de l'élitisme avec un autre processus de sélection. On a ainsi la garantie de garder les meilleurs individus tout en gardant l'avantage d'un processus de sélection aléatoire et d'une diversité de la population, caractéristique importante lors de l'utilisation d'algorithmes génétiques.

Une autre alternative consiste à effectuer la sélection sous forme de tournois. Il en existe plusieurs formes. Une des plus courantes est la suivante. Elle consiste à sélectionner deux individus aléatoirement, à ensuite comparer leur fitness et enfin, à attribuer une probabilité $p > 0.5$ d'être choisi à l'individu qui a le meilleur fitness et $1 - p$ à l'autre. Un autre type de tournoi est le tournoi de Deb [9]. Celui-ci permet de gérer les problèmes avec contraintes. Son principe est le suivant. Lorsque deux individus s'affrontent, l'individu admissible est préféré à l'individu non admissible. S'ils sont tous les deux admissibles, celui qui possède le meilleur fitness est préféré. S'ils sont tous les deux non admissibles, celui qui viole le moins les contraintes est préféré.

1.4.4 Reproduction

La reproduction permet de produire, à partir de deux individus parents, un individu enfant dont l'information génétique est un croisement de celle de ses parents. Elle s'effectue grâce à une recombinaison. Il s'agit d'un croisement des chromosomes des parents. Il en existe différents types. Nous présentons ici les trois plus simples, à savoir la recombinaison à un point, la recombinaison à n points et la recombinaison uniforme.

La recombinaison à un point est assez simple. Un point de coupure est choisi aléatoirement sur les chromosomes parents, séparant ainsi chaque chromosome en deux parties. L'enfant est composé de la première partie du chromosome du premier pa-

rent et de la seconde partie du chromosome du second parent. La recombinaison à n points est une généralisation de la recombinaison à un point. On choisit aléatoirement n points de coupure plutôt qu'un. On obtient donc $n+1$ morceaux de chromosomes. On échange un morceau sur deux des parents afin d'obtenir l'enfant. La recombinaison uniforme consiste à permuter chacun des gènes des chromosomes des parents avec une probabilité $\frac{1}{2}$. Une illustration des trois variantes présentées ci-dessus est fournie à la FIGURE 1.4.

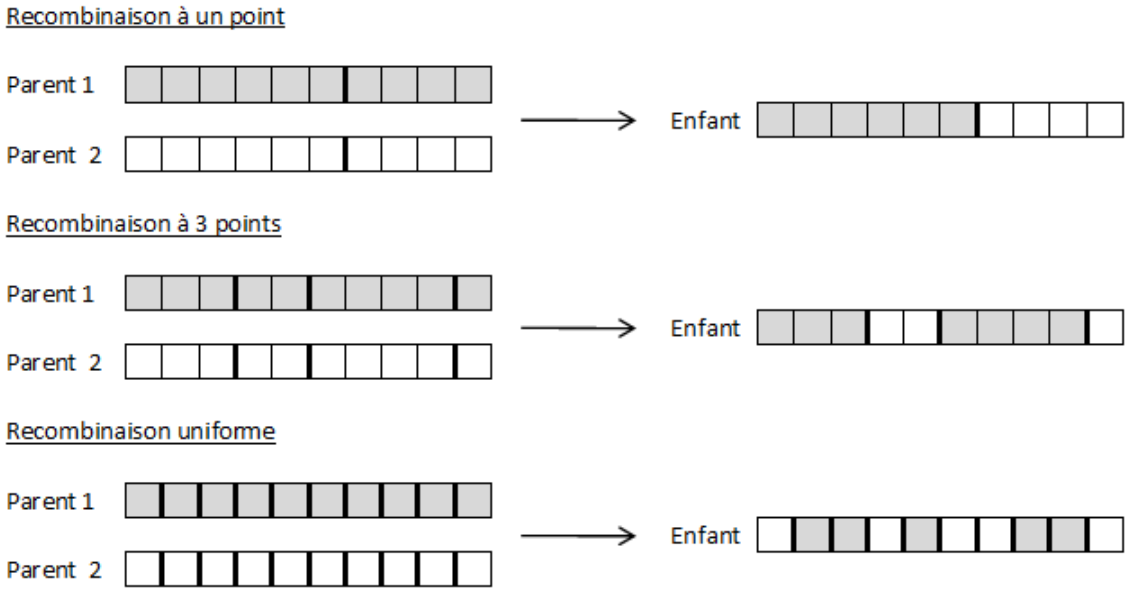


FIGURE 1.4 – Illustration des différents types de recombinaison

1.4.5 Mutation

Dans la nature, la mutation de certains gènes a lieu de façon très rare. Ce sont ces mutations qui permettent le mécanisme de l'évolution. De même, pour les algorithmes génétiques, chaque individu a une certaine probabilité d'être muté, généralement moins de dix pour cent. Ces mutations sont toutefois importantes car elles permettent de garantir une certaine diversité et permettent de s'échapper d'un optimum local. De nouveau, il existe différentes méthodes pour muter un individu. Nous nous contentons dans notre cas d'expliquer la plus simple. Celle-ci consiste à changer aléatoirement la valeur d'un gène d'un individu. Une illustration est fournie à la FIGURE 1.5.

FIGURE 1.5 – Illustration de la mutation d'un individu

1.4.6 Population suivante

Lorsque les opérateurs de sélection, de reproduction et de mutation ont été appliqués, nous disposons de trois populations : la population actuelle, la population issue de la reproduction ainsi que la population issue de la mutation. Parmi celles-ci, il faut choisir les individus qui constitueront la population de la génération suivante. Il existe différentes possibilités pour effectuer ce choix. On peut par exemple choisir de remplacer tous les individus de la population actuelle. Une alternative consiste à garder les n meilleurs et à remplacer les autres. Une autre alternative est de remplacer les n individus qui possèdent le moins bon fitness ou encore de remplacer n individus au hasard. Il est également possible de réaliser un tournoi entre les individus de la population actuelle et ceux des populations issues de la reproduction et de la mutation. Les vainqueurs constituent la population de la génération suivante.

Chapitre 2

Présentation de Cenaero et de Minamo

Ce chapitre se divise en deux parties. Nous commençons par décrire le centre de recherche Cenaero avec lequel ce mémoire est réalisé. Par la suite, nous expliquons le fonctionnement du logiciel MINAMO. Il s'agit du logiciel d'optimisation de Cenaero dans lequel les méthodes qui seront développées dans ce mémoire seront implémentées et analysées. Dans ce chapitre, nous nous basons sur les références [4], [8] et [23].

2.1 Le centre de recherche Cenaero

Cenaero [1] est un centre de recherche appliquée qui a vu le jour en mars 2002. Il est situé dans l'aéropole de Gosselies, en Belgique. Il a pour but de développer et de fournir des méthodologies et des outils de simulation numérique de haute-fidélité afin d'accompagner les sociétés innovantes dans la conception de produits performants.

Aujourd'hui, Cenaero emploie une soixantaine de personnes (dont la moitié avec thèse de doctorat) impliquées dans différents secteurs : aéronautique (propulsion), énergie et bâtiment, transport (terrestre, automobile, ferroviaire) et biomédical. Différents logiciels sont également développés à Cenaero : ARGO pour la simulation en mécanique des fluides, MINAMO pour la conception multi-disciplinaire et MORFEO, pour la fabrication virtuelle. Le centre de recherche est le partenaire privilégié de grands groupes tels que le groupe Safran avec lequel il collabore depuis 2007 et apporte aussi un soutien concret aux entreprises et PME en région wallonne. Ainsi, Cenaero travaille, entres autres, avec SABCA et SONACA et compte, parmi ses clients, des industries telles qu'ArcelorMittal et Caterpillar.

Cenaero possède également des infrastructures à la pointe de la technologie parmi lesquelles on retrouve le super-calculateur *Tier-1* qui a subi récemment une extension considérable et qui compte à présent plus de 11 500 coeurs. Il est donc le super-

1. Acronyme pour *centre d'excellence en recherche aéronautique.*

calculateur le plus important en Fédération Wallonie-Bruxelles. Dans son rapport annuel de 2013 [4], Cenaero mentionne l'importance de cette extension :

> *« Shared with the researchers of the Fédération Wallonie-Bruxelles, this infrastructure will allow this scientific community to further develop their projects and expertise in numerical simulation but further also demonstrate the relevance and applicability of their work at the European level. »*

Cette infrastructure est également mise à disposition des universités membres du Consortium des Équipements de Calcul intensif, à savoir l'UCL, l'ULB, l'ULg, l'UMons et l'UNamur.

2.2 Le logiciel Minamo

De nos jours, des centres de recherche tels que Cenaero ou plus généralement le monde de l'industrie utilisent de plus en plus de simulations de haute-fidélité. L'utilisation de celles-ci a été rendue possible grâce aux énormes progrès réalisés dans le domaine informatique ces dernières années. Ces simulations sont très coûteuses en temps. En effet, elles peuvent nécessiter plusieurs heures de calculs pour obtenir le résultat d'une seule simulation, et cela même si elles sont effectuées sur des super-ordinateurs tels que ceux utilisés par Cenaero. De plus, ces simulations, que l'on peut assimiler à l'évaluation d'une fonction, ne possèdent pas toujours une expression analytique connue. Ce sont ce qu'on appelle des fonctions « black-box ». Autrement dit, pour ces fonctions, la seule information connue est leur évaluation en certains points. Dès lors, la résolution de problèmes d'optimisation basés sur ces fonctions n'est pas des plus simples. En effet, tous les algorithmes d'optimisation qui ont recours à la dérivée ou au gradient ne peuvent être choisis car cette information est indisponible ou trop coûteuse à calculer. Une alternative est d'utiliser les algorithmes génétiques car ceux-ci ne nécessitent que l'évaluation de la fonction objectif en certains points. L'algorithme d'optimisation implémenté dans le logiciel Minamo se base notamment sur un algorithme génétique avec un codage de données réelles. Par convention, celui-ci est nommé algorithme évolutionnaire. Par la suite, les deux appellations seront indifféremment utilisées pour mentionner l'algorithme d'optimisation de Minamo.

Comme nous l'avons vu dans la Section 1.4, les algorithmes génétiques nécessitent un grand nombre d'évaluations de la fonction objectif. Le coût de celles-ci pouvant être relativement élevé, une approximation de la fonction objectif, appelée *méta-modèle* (ou modèle de substitution) est généralement utilisée plutôt que la fonction elle-même. On parle alors d'*optimisation assistée par méta-modèles*. Dans cette section, nous décrivons le fonctionnement général de l'algorithme d'optimisation assistée par méta-modèles de Minamo et nous analysons par la suite les différentes étapes de celui-ci. Notons que le logiciel Minamo permet également de travailler sur la fonction objectif elle-même, sans utiliser d'approximation. On parle alors d'*optimisation pure*. Dans ce cas, l'algorithme de Minamo applique simplement, à la fonction objectif, un algorithme génétique tel que ceux décrits à la Section 1.4. Néanmoins, il est à noter qu'au vu des applications

industrielles étudiées au sein de Cenaero, c'est l'algorithme évolutionnaire assisté par méta-modèles qui est le plus utilisé et qui a démontré la force de MINAMO. En effet, ces dernières années, l'investissement dans le développement de nouvelles techniques dans MINAMO a été principalement focalisé sur cette stratégie assistée par méta-modèles ; elle est dès lors la plus aboutie, comparativement à l'algorithme évolutionnaire pur.

2.2.1 Principe de base

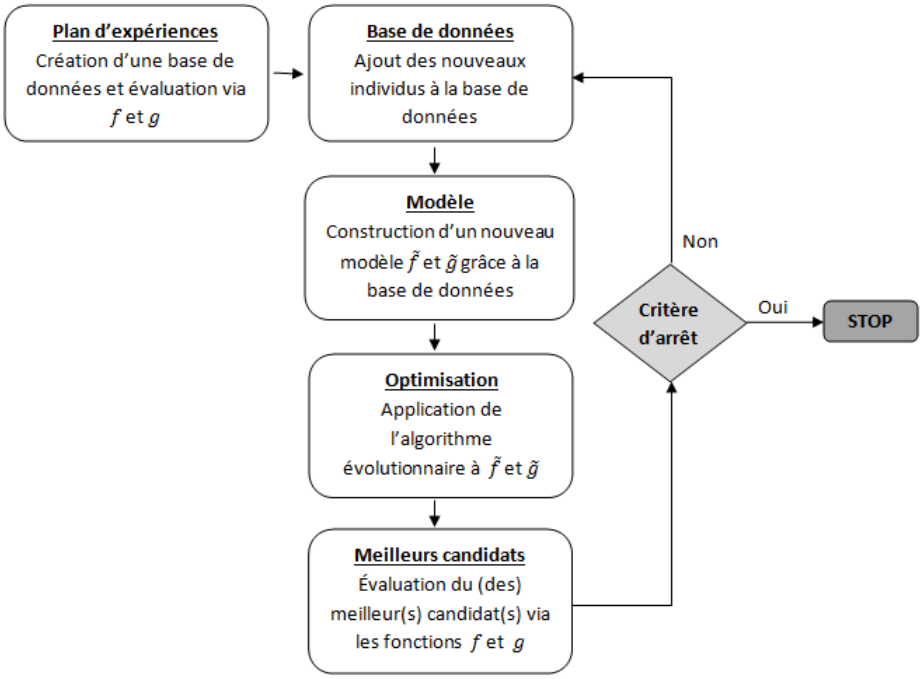


FIGURE 2.1 – Schéma de l'algorithme de MINAMO

Les différentes étapes de l'algorithme d'optimisation assistée par méta-modèles de MINAMO sont illustrées à la FIGURE 2.1. Il faut tout d'abord construire un *plan d'expériences*, plus communément appelé DoE (Design of Experiments). Cette première étape consiste à générer aléatoirement un ensemble de points suffisamment bien répartis dans l'espace. Dans le contexte des algorithmes évolutionnaires, cet ensemble de points est appelé population d'individus. La fonction objectif et les contraintes, que nous notons respectivement f et g, sont par la suite évaluées en chacun de ces individus. Ces informations constituent une *base de données* qui est utilisée afin de construire des méta-modèles $\tilde{f}$ et $\tilde{g}$ de la fonction objectif et des contraintes qui pourront être évalués à faible coût. Un algorithme évolutionnaire est ensuite appliqué au nouveau problème d'optimisation dont la fonction objectif et les contraintes sont $\tilde{f}$ et $\tilde{g}$. Le ou les meilleurs individus obtenus sont alors évalués au moyen de la fonction f et des contraintes g et sont ajoutés à la base de données. L'enrichissement de cette dernière permet d'améliorer

la qualité des méta-modèles qui deviendront, dès lors, plus précis d'itération en itération (processus dit *online*). Lors de chaque itération, nous procédons donc à l'optimisation de nouveaux méta-modèles ainsi qu'à l'évaluation et à l'ajout à la base de données du ou des meilleurs individus. Ce processus est répété jusqu'à atteindre le critère d'arrêt. Il s'agit généralement d'un nombre d'itérations fixé au préalable. Nous les appelons *itérations de conception*.

2.2.2 Plan d'expériences

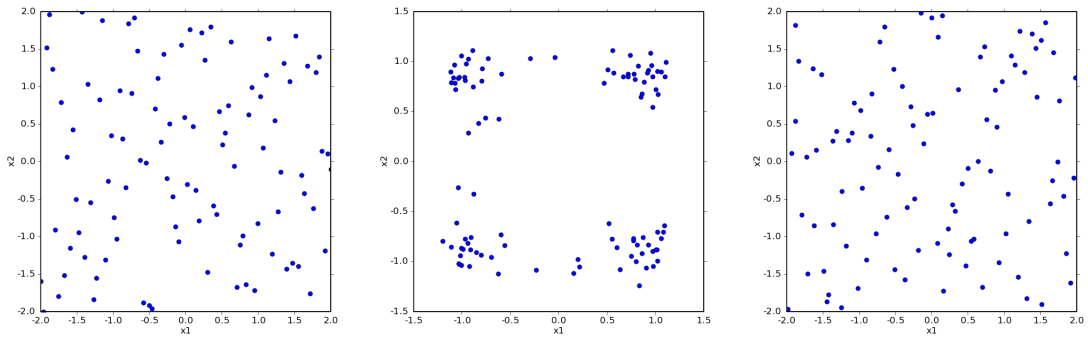
Cette première étape de l'algorithme consiste à générer une base de données constituée d'un certain nombre de points qui remplissent au mieux l'espace. Cette étape est très importante car la qualité du méta-modèle construit va dépendre de la façon dont les points ont été générés. L'idéal est d'obtenir le méta-modèle le plus précis possible en ayant recours à aussi peu d'évaluations de la fonction f et des contraintes g que possible. Nous distinguons deux catégories de plans d'expériences. Certaines méthodes sont dites *a priori*, c'est-à-dire qu'on n'utilise aucune information concernant la fonction f pour générer les points. Trois méthodes de ce type sont implémentées dans Minamo : *Latin Hypercube Sampling* (*LHS*), *Centroidal Voronoi Tessellation* (*CVT*) et *Latinized CVT* (*LCVT*) [24]. D'autres méthodes, que l'on appelle *a posteriori*, utilisent l'information de la fonction f pour générer les points de façon plus adéquate (*DoE auto-adaptatif*). Dans le cadre de notre travail, nous nous intéressons uniquement aux méthodes a priori.

La première méthode, le LHS, est assez simple. Il s'agit d'un échantillonnage en carré latin. En deux dimensions, un carré latin est une grille contenant l lignes et l colonnes et qui est composée de l^2 carrés de même taille. Un ensemble de l points est réparti dans la grille de telle sorte que chaque ligne et chaque colonne contienne exactement un point. En dimension $n > 2$, on parle de la notion d'hypercube latin. Celle-ci est similaire à la notion de carré latin, il suffit de généraliser le concept à un plus grand nombre de dimensions. L'avantage de cette méthode se situe dans la simplicité du procédé. Cependant, c'est aussi la simplicité du procédé qui est responsable d'un gros désavantage. En effet, il autorise certains cas où les points ne recouvrent pas bien tout l'espace. L'exemple le plus extrême est le cas où tous les points sont situés sur une diagonale du carré. Un échantillonnage en carré latin est illustré à la Figure 2.2a.

La deuxième méthode, le CVT, est basée sur une partition de l'espace en régions qui sont définies à partir de générateurs. Ceux-ci sont des points répartis dans l'espace. À chaque générateur correspond une région. Celle-ci est l'ensemble des points qui sont plus proches de ce générateur que de n'importe quel autre générateur. Contrairement au LHS, cette technique permet de générer des points qui recouvrent mieux tout l'espace à l'exception des bords. Toutefois, lorsqu'on projette les points dans un espace de dimension plus petite, ils sont répartis en groupes, ce qui signifie qu'ils ne remplissent pas bien tout l'espace, voir Figure 2.2b.

Une solution pour rassembler les avantages des deux méthodes ci-dessus sans en avoir les inconvénients est de les combiner. Cette méthode porte le nom de Latinized CVT (LCVT) car il s'agit simplement d'une latinisation d'un ensemble de points générés par

la méthode CVT. Autrement dit, on commence par générer un ensemble de points avec la méthode CVT et ensuite, on opère une latinisation de ces points. Le principe de la latinisation est de transformer l'ensemble de points de telle sorte qu'il vérifie la propriété des hypercubes latins. On évite ainsi le regroupement des points lorsqu'on effectue une projection tout en gardant une bonne répartition dans l'espace, voir FIGURE 2.2c. Cette méthode est la plus utilisée dans MINAMO.



(a) LHS (b) CVT (c) LCVT

FIGURE 2.2 – Projection sur le plan (x_1, x_2) de 100 points d'un espace à 10 dimensions - Source : [23]

2.2.3 Méta-modèles

Comme nous l'avons déjà mentionné précédemment, MINAMO est utilisé pour optimiser des fonctions dont l'évaluation requiert un grand temps de calcul. Il est alors moins coûteux de travailler sur des méta-modèles plutôt que sur les fonctions elles-mêmes. Ceux-ci doivent être relativement précis tout en nécessitant peu d'évaluations de fonctions. Les modèles d'interpolation polynomiale ne sont pas très performants pour approximer des fonctions multimodales, c'est-à-dire des fonctions qui possèdent des optima locaux autres que l'optimum global. Les différentes techniques implémentées dans MINAMO sont des interpolations non linéaires. Il s'agit de *fonctions à base radiale* (*RBF*), de *modèles de Kriging* et de *machines à vecteurs de supports* (*SVM*) [23].

2.2.4 Algorithme évolutionnaire

Dans MINAMO, la phase d'optimisation se fait au moyen d'un algorithme génétique du même type que ceux présentés à la Section 1.4. Cet algorithme est appliqué à différentes reprises. En effet, plutôt que d'optimiser directement la fonction objectif, on optimise un méta-modèle. Après l'optimisation de celui-ci, on l'enrichit, ce qui nous permet d'obtenir un méta-modèle plus précis qu'on optimise de nouveau et ainsi de suite.

Comme nous l'avons vu à la Section 1.4, plusieurs choix sont possibles lors des différentes étapes d'un tel algorithme. Le mécanisme utilisé lors de l'étape de sélection est le tournoi de Deb. Des opérateurs de recombinaison et de mutation sont ensuite appliqués. Pour ces étapes, le codage des données joue un rôle important. Dans Minamo, le codage réel a été choisi car il permet de représenter un individu de façon simple et compacte. Finalement, un processus d'élitisme est appliqué à chaque génération afin d'ajouter les meilleurs individus à la population suivante.

Chapitre 3

État de l'art des algorithmes évolutionnaires distribués

Les algorithmes évolutionnaires, dont font partie les algorithmes génétiques présentés à la Section 1.4, sont des algorithmes d'optimisation qui fournissent un optimum global au moyen d'un processus stochastique. Comme expliqué au Chapitre 1, leur fonctionnement s'inspire de la théorie de l'évolution de Charles Darwin selon laquelle ce sont les espèces les mieux adaptées à leur environnement qui se perpétuent tandis que les autres disparaissent. Ces algorithmes se montrent efficaces et performants pour de nombreux problèmes. Toutefois, lorsqu'on traite des problèmes complexes de grande dimension (plus de 100 variables), il est relativement difficile d'en approcher la solution et le temps de calcul nécessaire afin de les résoudre peut vite devenir très, voire trop grand. Une solution afin d'y remédier consiste à diviser le problème en sous-problèmes. Pour ce faire, la population, constituée de solutions potentielles, est divisée en sous-populations. L'optimisation est ensuite opérée sur chacune des sous-populations, celles-ci *coopérant* entre elles. On parle alors d'*algorithmes évolutionnaires distribués, coopératifs*, ou encore de *co-évolution coopérative*. Ceux-ci permettent de résoudre des problèmes dont le nombre de variables peut monter jusque 1 000. Mitchell A. Potter [22] est considéré comme le fondateur de ces méthodes. Dans ce chapitre, nous abordons les étapes de décomposition, de coopération et d'optimisation de ces algorithmes.

3.1 La décomposition

L'étape majeure des algorithmes évolutionnaires distribués consiste à diviser, de manière adéquate, la population en sous-populations. Supposons que nous souhaitons minimiser une fonction $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$. Le principe de l'étape de décomposition consiste à séparer l'ensemble des n variables $(x_1, \ldots, x_n)$ en m sous-ensembles. À chaque sous-ensemble de variables, correspond une sous-population. Il s'agit des individus caractérisés par les variables du sous-ensemble en question. Chaque sous-population est donc composée des individus de la population initiale, ces individus n'étant caractérisés

que par un sous-ensemble de variables. À l'Exemple 3.1, nous illustrons cette décomposition d'une population en sous-populations.

Exemple 3.1
Soit une population de dix individus caractérisés par six variables $x_1, x_2, \ldots, x_6$,

$$\begin{aligned} I^1 &= (x_1^1, x_2^1, x_3^1, x_4^1, x_5^1, x_6^1) \\ I^2 &= (x_1^2, x_2^2, x_3^2, x_4^2, x_5^2, x_6^2) \\ &\vdots \\ I^{10} &= (x_1^{10}, x_2^{10}, x_3^{10}, x_4^{10}, x_5^{10}, x_6^{10}). \end{aligned}$$

Nous choisissons de diviser la population en trois sous-populations et de grouper les variables 1 *et* 5, *les variables* 2 *et* 3, *ainsi que les variables* 4 *et* 6. *Les trois sous-populations obtenues sont alors données dans le tableau ci-dessous.*

sous-population (1,5)	*sous-population (2,3)*	*sous-population (4,6)*
$I_{1,5}^1 = (x_1^1, x_5^1)$	$I_{2,3}^1 = (x_2^1, x_3^1)$	$I_{4,6}^1 = (x_4^1, x_6^1)$
$I_{1,5}^2 = (x_1^2, x_5^2)$	$I_{2,3}^2 = (x_2^2, x_3^2)$	$I_{4,6}^2 = (x_4^2, x_6^2)$
$\vdots$	$\vdots$	$\vdots$
$I_{1,5}^{10} = (x_1^{10}, x_5^{10})$	$I_{2,3}^{10} = (x_2^{10}, x_3^{10})$	$I_{4,6}^{10} = (x_4^{10}, x_6^{10})$

Une fois la décomposition opérée, un algorithme évolutionnaire classique est appliqué à chacune de ces sous-populations. Bien sûr, la qualité de la solution obtenue dépend grandement de la séparation de l'ensemble des variables en sous-ensembles. Dès lors, dans le but d'effectuer cette séparation de manière adéquate, nous commençons par introduire le concept de *fonction séparable* [26].

Définition 3.1
La fonction $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ *est une fonction séparable si,* $\forall\ k \in \{1, 2, \ldots, n\}$ *:*

$$\left[\left. \begin{array}{ll} x \in D & x = (x_1, \ldots, x_k, \ldots, x_n) \\ x' \in D & x' = (x_1, \ldots, x'_k, \ldots, x_n) \end{array} \right\} \Rightarrow f(x) < f(x') \right]$$

implique

$$\left[\left. \begin{array}{ll} \forall\ y \in D & y = (y_1, \ldots, x_k, \ldots, y_n) \\ \forall\ y' \in D & y' = (y_1, \ldots, x'_k, \ldots, y_n) \end{array} \right\} \Rightarrow f(y) < f(y') \right].$$

Sinon, elle est non séparable.

Sur base de cette définition, on dit qu'une fonction est séparable si l'influence de chacune des variables sur la valeur du fitness ne dépend que d'elle-même. Les variables sont donc indépendantes les unes des autres. Dans le cas contraire, la fonction est non séparable, ce qui signifie que certaines variables interagissent entre elles et sont donc dépendantes les unes des autres.

Exemple 3.2

1. *La fonction* $f : \mathbb{R}^2 \longrightarrow \mathbb{R} : (x_1, x_2) \mapsto f(x_1, x_2) = x_1^2 + x_2^2$ *est séparable.*

   Preuve
   Soient $k = 1$, (x_1, x_2), $(x_1', x_2) \in \mathbb{R}^2$ *tels que* $f(x_1, x_2) < f(x_1', x_2)$, *il s'en suit que*

   $$x_1^2 < {x_1'}^2. \tag{3.1}$$

   Dès lors, $\forall\ (x_1, y)$, $(x_1', y) \in \mathbb{R}^2$, *on a*

   $$f(x_1, y) = x_1^2 + y^2 < {x_1'}^2 + y^2 = f(x_1', y)$$

   où l'inégalité stricte résulte de l'équation (3.1). Le cas où $k = 2$ *se traite de façon similaire.*

   ■

2. *La fonction* $f : \mathbb{R}^2 \longrightarrow \mathbb{R} : (x_1, x_2) \mapsto f(x_1, x_2) = x_1^2 + (x_1 + x_2)^2$ *est non séparable car*

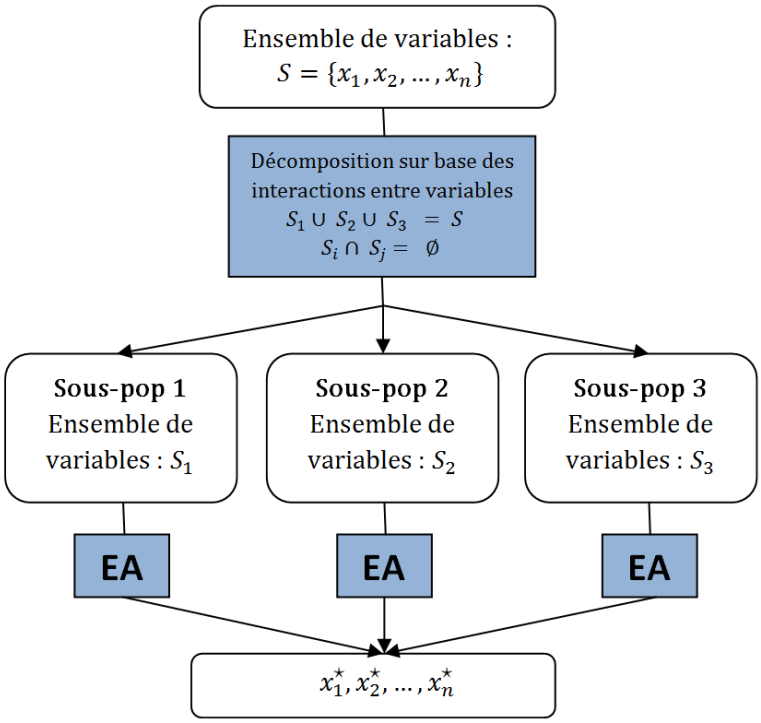
   $$f(0,1) < f(0,2) \qquad \not\Rightarrow \qquad \forall\ y \in \mathbb{R} \qquad f(y,1) < f(y,2).$$

   En effet,

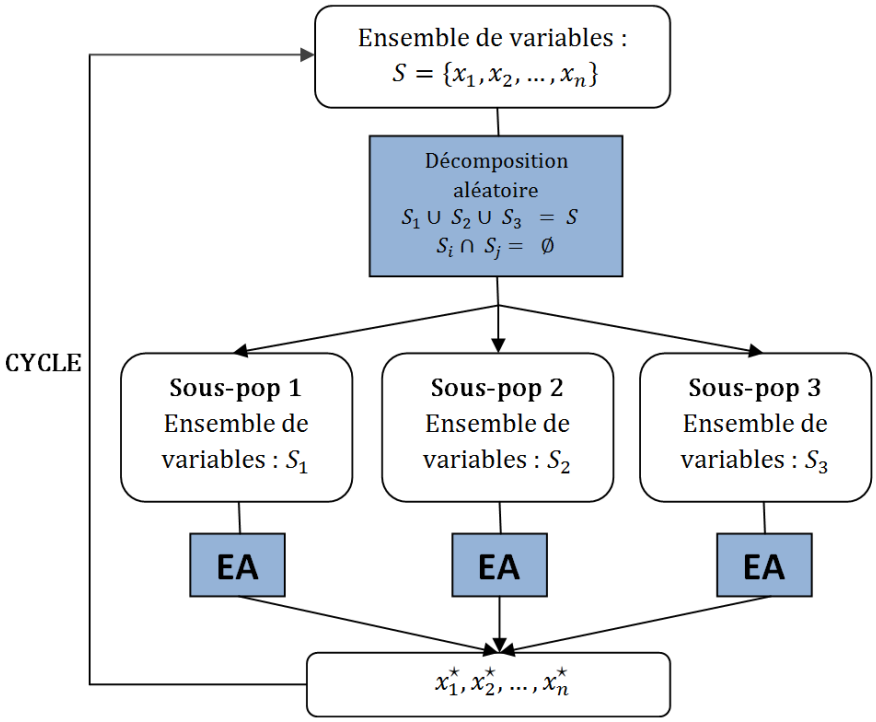
   $$f(-2,1) > f(-2,2).$$

Certains algorithmes évolutionnaires distribués, tels que ceux proposés par Omidvar, Li, Mei et Yao [20], utilisent cette notion de variables dépendantes afin de grouper, dans une même sous-population, les variables qui interagissent le plus entre elles. De cette façon, il y a peu ou pas d'interactions entre des variables appartenant à des sous-populations différentes. Le mécanisme, dont une illustration est proposée à la FIGURE 3.1a, consiste alors à grouper les variables une fois pour toutes en début d'algorithme et d'ensuite optimiser chacune des sous-populations grâce à un algorithme évolutionnaire classique. Ces algorithmes proposés par Omidvar, Li, Mei et Yao ont été testés et comparés à d'autres algorithmes sur des fonctions test. Il en ressort qu'ils fournissent de bons résultats et que l'étape de décomposition, bien que nécessitant un certain coût en termes de ressources, permet une meilleure convergence que celle fournie par d'autres algorithmes.

Yang, Tang et Yao [26] proposent, quant à eux, un mécanisme de décompositions aléatoires. Nous implémenterons celui-ci dans MINAMO (voir Chapitre 5). Son principe est assez simple. Il consiste à séparer aléatoirement l'ensemble des n variables $(x_1, \ldots, x_n)$ en m sous-ensembles de taille s de telle sorte que $n = m \times s$. À chaque sous-ensemble de variables, correspond une sous-population. Un algorithme évolutionnaire classique est appliqué à chacune de ces sous-populations. Ensuite, une nouvelle décomposition aléatoire est effectuée. Un algorithme évolutionnaire est de nouveau appliqué à chacune des sous-populations et ainsi de suite jusqu'à atteindre un critère d'arrêt. Il s'agit généralement d'un nombre de cycles fixé au préalable, un cycle étant une exécution de l'algorithme évolutionnaire pour une décomposition aléatoire. Une illustration de ce mécanisme est proposée à la FIGURE 3.1b. L'efficacité de ces décompositions aléatoires est garantie par le théorème suivant.



(a) Une seule décomposition (Omidvar, Li, Mei, Yao)



(b) Plusieurs décompositions aléatoires (Yang, Tang, Yao)

Figure 3.1 – Différents types d'algorithmes évolutionnaires distribués

Théorème 3.1
La probabilité de grouper deux variables dépendantes dans le même sous-ensemble pour au moins k cycles est :

$$P_k = \sum_{r=k}^{N} \binom{N}{r} \left(\frac{1}{m}\right)^r \left(1 - \frac{1}{m}\right)^{N-r},$$

où N est le nombre total de cycles et m le nombre de sous-ensembles.

Démonstration
Soient deux variables x_i et x_j, lors de chaque cycle, la probabilité p de grouper ces deux variables est de $\frac{1}{m}$ car il y a m sous-ensembles. Dès lors, les groupements de chaque cycle étant indépendants les uns des autres, la probabilité p_r de grouper ces deux variables lors d'exactement r cycles suit une distribution binomiale et vaut

$$p_r = \binom{N}{r} p^r (1-p)^{N-r} = \binom{N}{r} \left(\frac{1}{m}\right)^r \left(1 - \frac{1}{m}\right)^{N-r}.$$

La probabilité P_k de grouper ces deux variables pour au moins k cycles est donc donnée par

$$P_k = \sum_{r=k}^{N} \binom{N}{r} \left(\frac{1}{m}\right)^r \left(1 - \frac{1}{m}\right)^{N-r}.$$

■

Par exemple, pour $N = 30$ et $m = 5$, (ce choix de paramètres sera notamment utilisé aux Sections 5.2.1 et 5.2.2 lors de la présentation des résultats), on a $P_1 = 0.9987$ et $P_2 = 0.9894$, ce qui signifie qu'on a de très grandes chances de grouper deux variables dépendantes dans un même sous-ensemble pour au moins deux cycles. Cela est d'autant plus remarquable qu'aucune information sur la fonction à optimiser n'est requise afin d'effectuer ces groupements.

Ce mécanisme de décompositions aléatoires est également étudié par Omidvar, Li, Yang et Yao dans [21]. Ils y proposent d'auto-adapter, en cours d'optimisation, le nombre de sous-populations. Lorsqu'aucune amélioration du fitness n'est obtenue d'un cycle à l'autre, un nouveau nombre de sous-populations est utilisé pour le cycle suivant. Ce dernier est choisi aléatoirement dans un ensemble définissant les nombres de sous-populations qui peuvent être choisis. Il a été montré que le nouvel algorithme ainsi obtenu fournit de meilleurs résultats qu'un algorithme où le nombre de sous-populations est fixé pour toute l'optimisation. Nous nous inspirerons de ce mécanisme d'auto-adaptation pour développer l'algorithme présenté au Chapitre 6.

3.2 La coopération

Lors de la phase d'optimisation des différentes sous-populations, celles-ci coopèrent. Pour ce faire, chacune des sous-populations est représentée par un individu dit *représentant*. Différents choix sont possibles. On peut simplement choisir, pour représentants,

le meilleur individu de chacune des sous-populations. Il est également possible de les choisir aléatoirement ou bien d'effectuer un processus stochastique ou déterministe sur base de leur fitness. Ces représentants sont utilisés lors de l'évaluation du fitness. En effet, chaque individu est caractérisé par s variables et, la fonction objectif f étant une fonction à n variables, doit être complété par $n - s$ variables pour être évalué. Ces variables sont celles des représentants des autres sous-populations. Ce processus d'évaluation est illustré à l'Exemple 3.3.

Exemple 3.3
Cet exemple constitue la suite de l'Exemple 3.1. Il consiste à illustrer l'évaluation du fitness d'un individu. Ainsi, le fitness de l'individu $I^2_{2,3} = (x^2_2, x^2_3)$ *est donné par*

$$f(x^\star_1, x^2_2, x^2_3, x^\star_4, x^\star_5, x^\star_6)$$

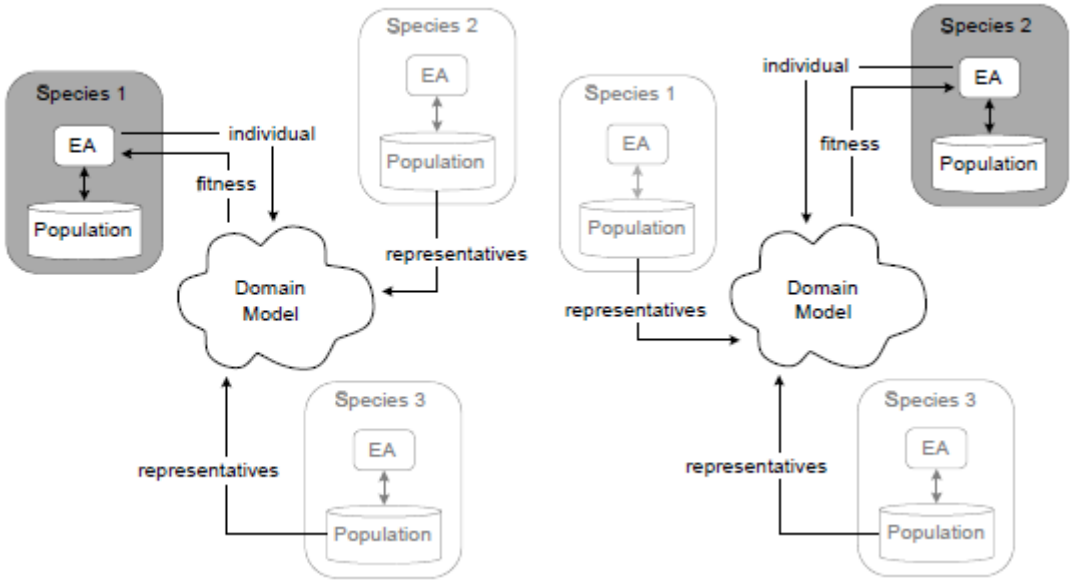
où $I^\star_{1,5} = (x^\star_1, x^\star_5)$ *et* $I^\star_{4,6} = (x^\star_4, x^\star_6)$ *sont les représentants respectifs des sous-populations* $(1, 5)$ *et* $(4, 6)$.

Une seconde illustration du mécanisme de coopération entre trois sous-populations est également fournie à la Figure 3.2. Sur celle-ci, le terme *Species* désigne ce que nous appelons sous-populations. La Figure 3.2a (resp. 3.2b et 3.2c) illustre l'évaluation des individus de la sous-population 1 (resp. 2 et 3). Ainsi, pour évaluer un individu de la sous-population 1, nous pouvons voir à la Figure 3.2a que le fitness (ici représenté par le *Domain Model*) est évalué à partir de cet individu et des représentants des sous-populations 2 et 3.

3.3 L'optimisation

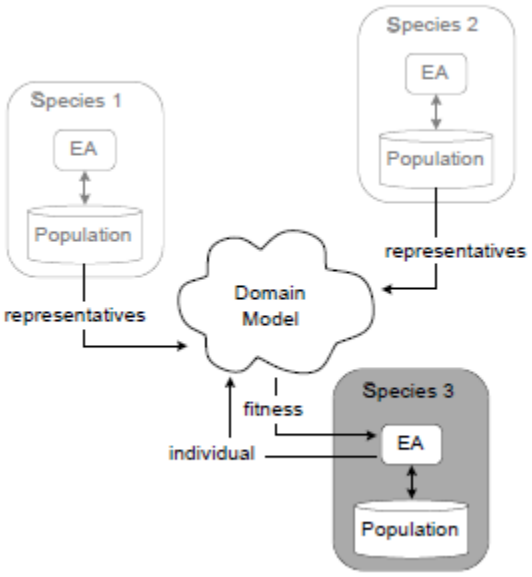
Les optimisations des différentes sous-populations sont réalisées par un algorithme évolutionnaire classique. Elles ne sont pas réalisées séquentiellement, c'est-à-dire les unes après les autres, mais parallèlement. En effet, les étapes de la première génération des algorithmes évolutionnaires (sélection, reproduction, mutation et population suivante) sont d'abord effectuées sur chacune des sous-populations. Les représentants de chaque sous-population sont ensuite mis-à-jour. Par après, les étapes de la deuxième génération ainsi que la mise-à-jour des représentants sont effectuées, etc. Ce mécanisme est proposé à l'Algorithme 3.1.

Dans cet algorithme, la coopération a lieu au cours des phases « Sélection », « Population suivante » et « Mise-à-jour des représentants ». En effet, des évaluations du fitness sont nécessaires lors de ces phases. Celles-ci sont effectuées grâce à la méthode décrite à la Section 3.2, c'est-à-dire que le fitness d'un individu d'une sous-population est évalué à partir de cet individu ainsi que des représentants des autres sous-populations.



(a) Sous-population 1

(b) Sous-population 2



(c) Sous-population 3

Figure 3.2 – Coopération entre trois sous-populations - Source : [22]

Algorithme 3.1 : Optimisation des sous-populations par l'algorithme évolutionnaire

```
pour chaque génération faire
    pour chaque sous-population faire
        Sélection ;
        Reproduction ;
        Mutation ;
        Population suivante ;
    fin
    Mise-à-jour des représentants ;
fin
```

Chapitre 4

Présentation des tests de validation

Ce chapitre a pour but de décrire, d'une part, l'ensemble des fonctions test utilisées pour valider l'implémentation de nos algorithmes évolutionnaires distribués que nous présentons dans les Chapitres 5 et 6. Nous en distinguons trois classes. D'abord les fonctions dites « classiques », qui sont des fonctions couramment utilisées dans le but de tester la performance d'algorithmes d'optimisation. Ensuite les fonctions transformées, qui sont des fonctions classiques auxquelles nous avons appliqué une transformation non linéaire à l'ensemble des variables, cela dans le but de casser la symétrie de ces fonctions. Enfin, les fonctions avec rotation qui sont des fonctions classiques auxquelles nous avons appliqué une rotation à l'ensemble des variables. Cette rotation permet de rendre ces fonctions non séparables. D'autre part, nous décrivons également dans ce chapitre les outils à l'aide desquels nous présentons nos résultats dans les chapitres suivants. Il s'agit de graphes de convergence et de tableaux statistiques.

4.1 Les fonctions « classiques »

Les fonctions décrites dans cette section sont issues de [27]. Nous en donnons l'expression analytique, l'optimum global ainsi que la valeur de la fonction en ce point. Nous fournissons également une représentation des fonctions ainsi que leurs courbes de niveau. De plus, les fonctions décrites dans cette section présentent des caractéristiques variées couvrant ainsi un large éventail de problèmes d'optimisation. Certaines sont unimodales (Elliptique, Quartique, etc.), d'autres sont multimodales (Generalized Penalized 1 et 2, Schwefel 2.26, etc.). Plusieurs d'entre elles sont séparables (Sphère, Rastrigin, etc.), d'autres sont non séparables (Schwefel 12, Rosenbrock, etc.). Nombreuses sont différentiables (Ackley, Griewank, etc.) tandis que d'autres sont non différentiables (Schwefel 2.21, Step, etc.). Enfin, l'ensemble de ces fonctions sont multi-dimensionnelles, autrement dit, il est possible, pour chacune d'entre elles, de choisir le nombre de variables. Cela nous permet de tester notre algorithme distribué sur des fonctions de dimension 10, 50 et 500.

• **Ackley**
L'expression analytique de la fonction Ackley est donnée par

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e,$$

où $e = \exp(1)$, n est le nombre de variables et $x \in [-32, 32]^n$. Elle est multimodale, son minimum global se trouve en

$$x_i^\star = 0, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.1. Un zoom autour de l'optimum global est également fourni à la FIGURE 4.2.

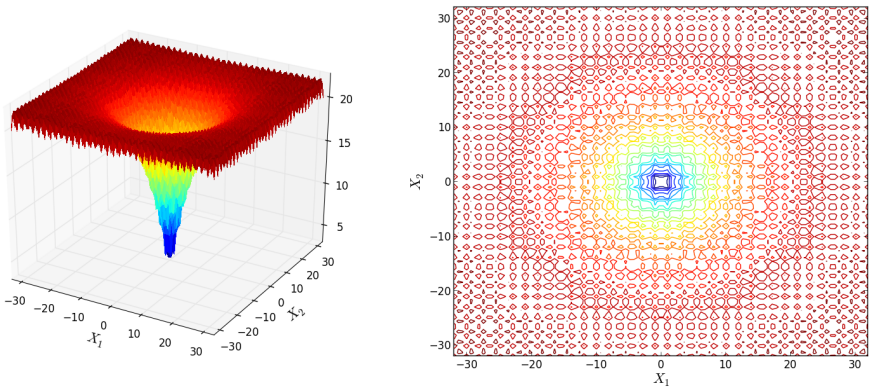


FIGURE 4.1 – Fonction Ackley : représentation et courbes de niveau

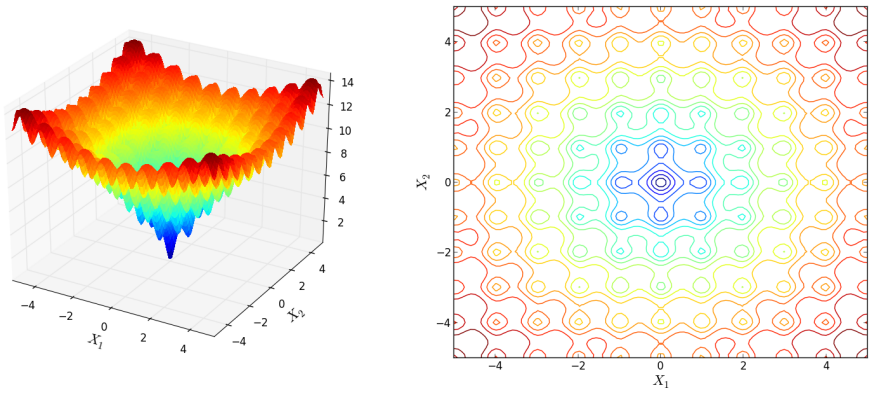


FIGURE 4.2 – Fonction Ackley : zoom autour de l'optimum global

• **Elliptique**
L'expression analytique de la fonction Elliptique est donnée par

$$f(x) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} x_i^2,$$

où n est le nombre de variables et $x \in [-100, 100]^n$. Elle possède un seul minimum en

$$x_i^\star = 0, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.3.

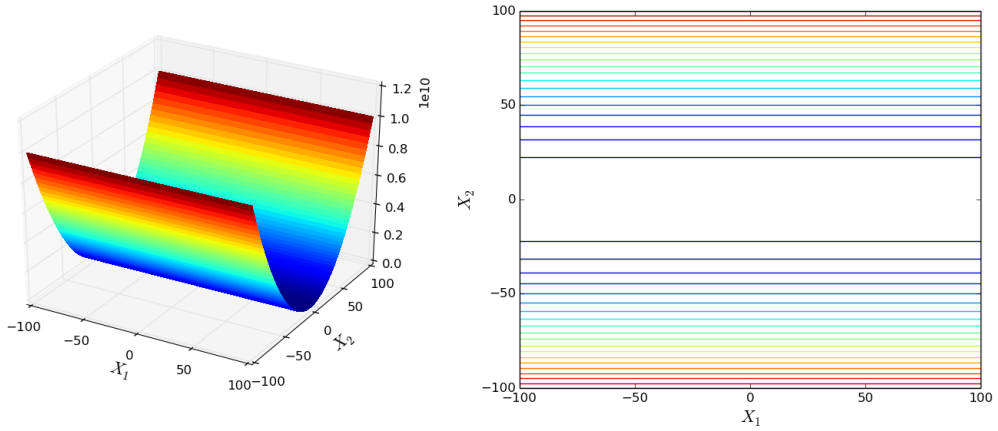


FIGURE 4.3 – Fonction Elliptique : représentation et courbes de niveau

• **Generalized Penalized 1**
L'expression analytique de la fonction Generalized Penalized 1 est donnée par

$$f(x) = \frac{\pi}{n}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right) + \sum_{i=1}^{n} u(x_i, 10, 100, 4),$$

où

$$y_i = 1 + \frac{1}{4}(x_i + 1), \qquad u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \leq x_i \leq a, \\ k(-x_i - a)^m, & x_i < -a, \end{cases}$$

n est le nombre de variables et $x \in [-50, 50]^n$. Elle est multimodale, son minimum global se trouve en

$$x_i^\star = -1, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.4. Un zoom autour de l'optimum global est également fourni à la FIGURE 4.5.

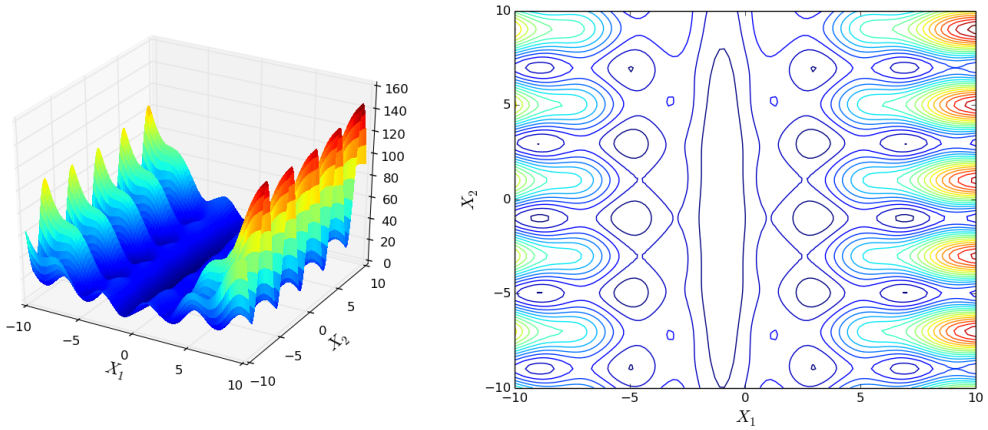


FIGURE 4.4 – Fonction Generalized Penalized 1 : représentation et courbes de niveau

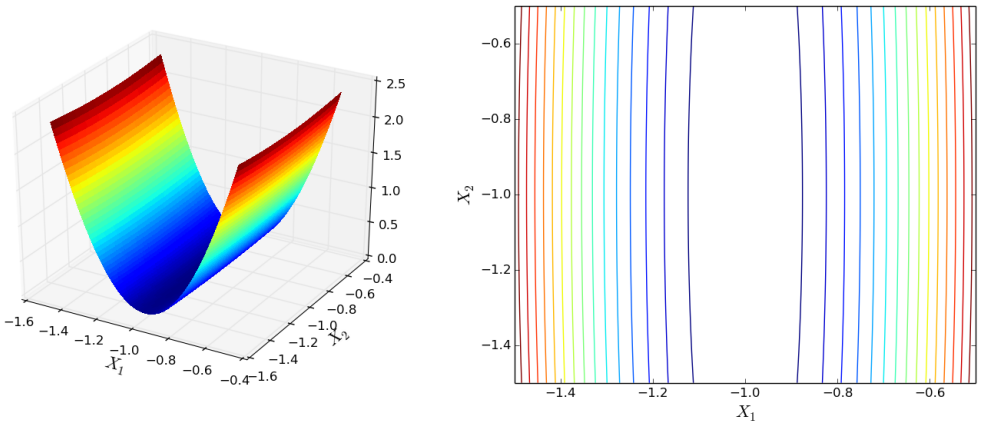


FIGURE 4.5 – Fonction Generalized Penalized 1 : zoom autour de l'optimum global

• **Generalized Penalized 2**

L'expression analytique de la fonction Generalized Penalized 2 est donnée par

$$\begin{aligned} f(x) &= 0.1\left(\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})] + (x_n-1)^2[1+\sin^2(3\pi x_n)]\right) \\ &+ \sum_{i=1}^{n} u(x_i, 5, 100, 4), \end{aligned}$$

où n est le nombre de variables et $x \in [-50, 50]^n$. Elle est multimodale, son minimum global se trouve en

$$x_i^\star = 1, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.6. Un zoom autour de l'optimum global est également fourni à la FIGURE 4.7.

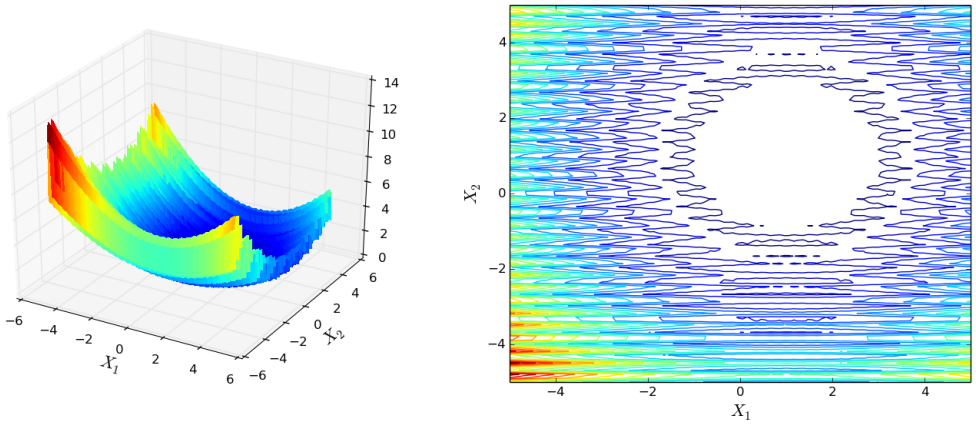


FIGURE 4.6 – Fonction Generalized Penalized 2 : représentation et courbes de niveau

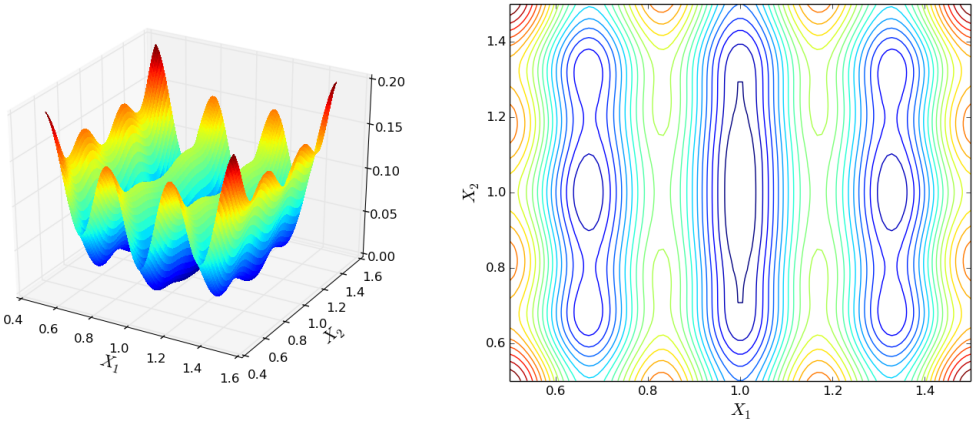


FIGURE 4.7 – Fonction Generalized Penalized 2 : zoom autour de l'optimum global

• **Griewank**

L'expression analytique de la fonction Griewank est donnée par

$$f(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1,$$

où n est le nombre de variables et $x \in [-600, 600]^n$. Elle est multimodale, son minimum global se trouve en

$$x_i^\star = 0, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.8. Un zoom autour de l'optimum global est également fourni à la FIGURE 4.9.

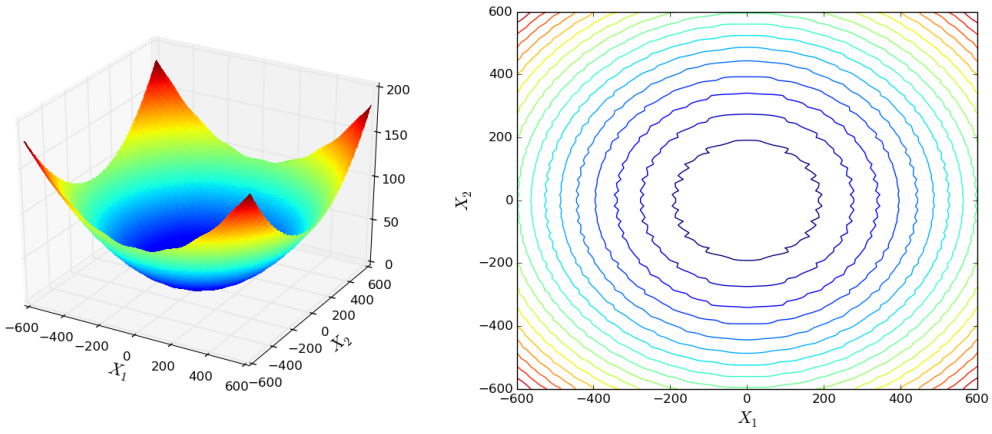


FIGURE 4.8 – Fonction Griewank : représentation et courbes de niveau

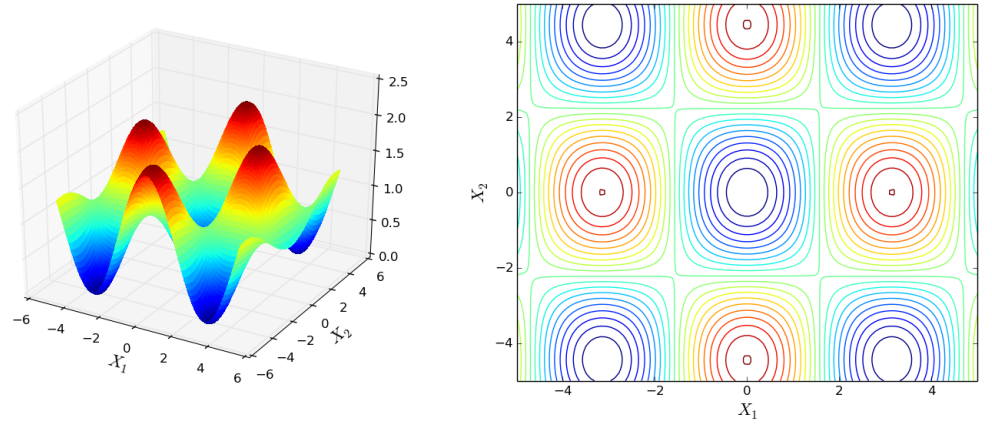


FIGURE 4.9 – Fonction Griewank : zoom autour de l'optimum global

• **Quartique**

L'expression analytique de la fonction Quartique est donnée par

$$f(x) = \sum_{i=1}^{n} i x_i^4 + random[0, 1[,$$

où $random[0, 1[$ désigne un nombre aléatoirement généré dans l'intervalle $[0, 1[$, n est le nombre de variables et $x \in [-1.28, 1.28]^n$. Elle possède un seul minimum en

$$x_i^\star = 0, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.10.

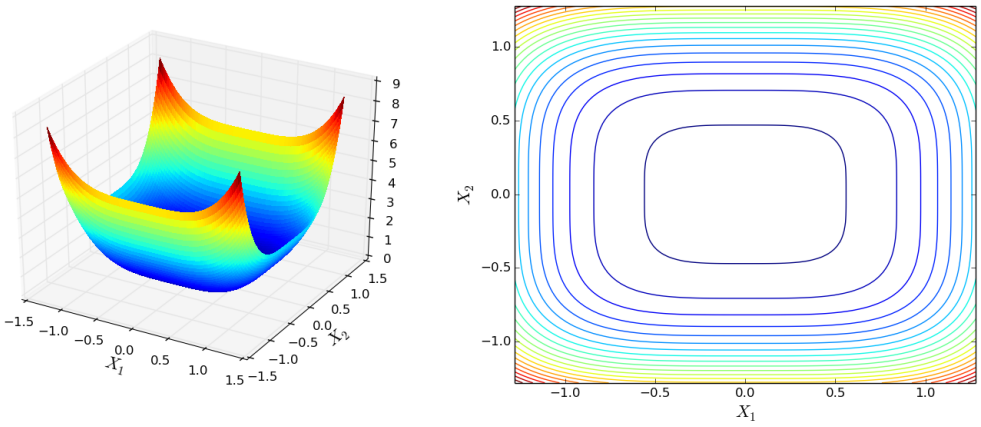


FIGURE 4.10 – Fonction Quartique : représentation et courbes de niveau

• **Rastrigin**

L'expression analytique de la fonction Rastrigin est donnée par

$$f(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10],$$

où n est le nombre de variables et $x \in [-5.12, 5.12]^n$. Elle est multimodale, son minimum global se trouve en

$$x_i^\star = 0, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.11.

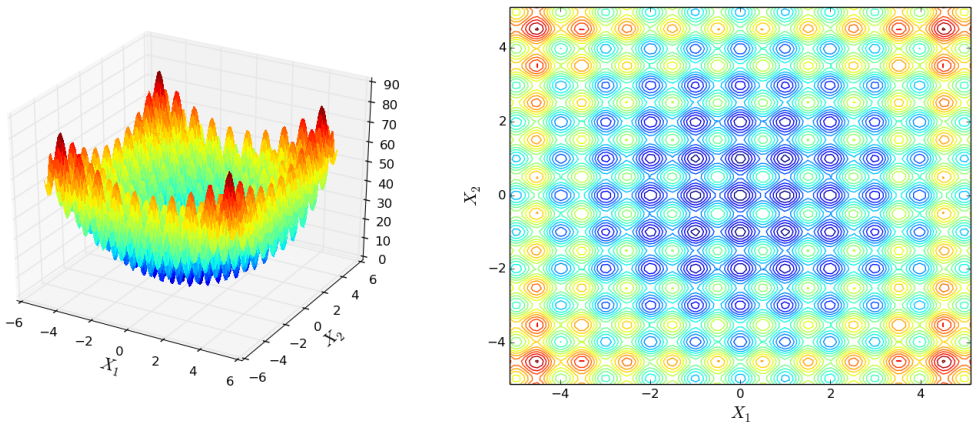


FIGURE 4.11 – Fonction Rastrigin : représentation et courbes de niveau

• **Rosenbrock**

L'expression analytique de la fonction Rosenbrock est donnée par

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2],$$

où n est le nombre de variables et $x \in [-30, 30]^n$. Elle est unimodale pour $n = 2$ et $n = 3$ mais devient multimodale pour $n \geq 4$. Son minimum global se trouve en

$$x_i^\star = 1, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.12. Un zoom autour de l'optimum global est également fourni à la FIGURE 4.13.

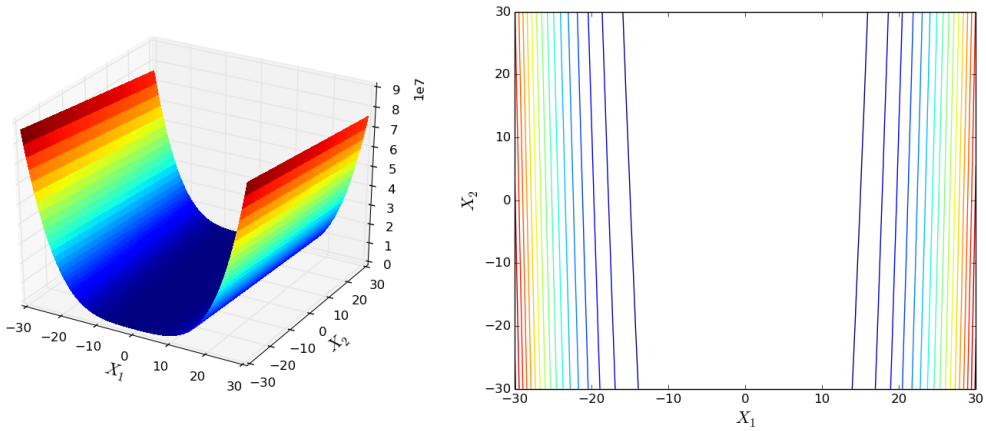


FIGURE 4.12 – Fonction Rosenbrock : représentation et courbes de niveau

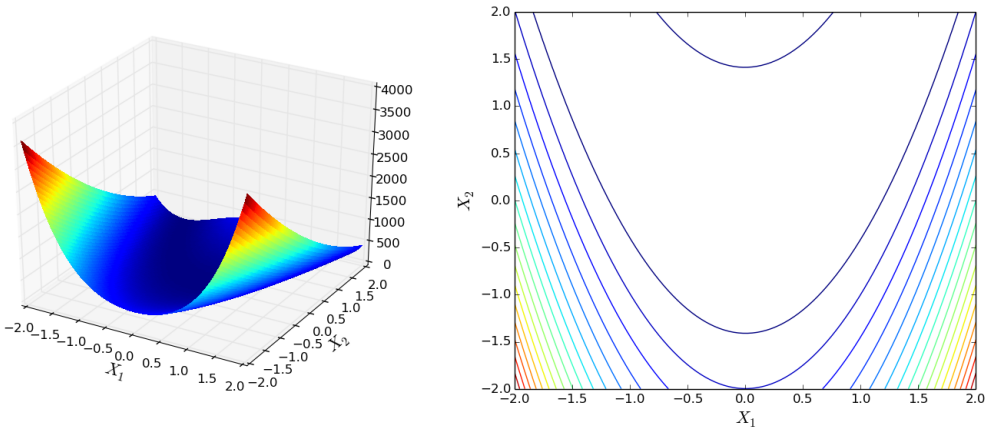


FIGURE 4.13 – Fonction Rosenbrock : zoom autour de l'optimum global

• **Schwefel 12**
L'expression analytique de la fonction Schwefel 12 est donnée par

$$f(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2,$$

où n est le nombre de variables et $x \in [-100, 100]^n$. Elle possède un seul minimum en

$$x_i^\star = 0, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.14.

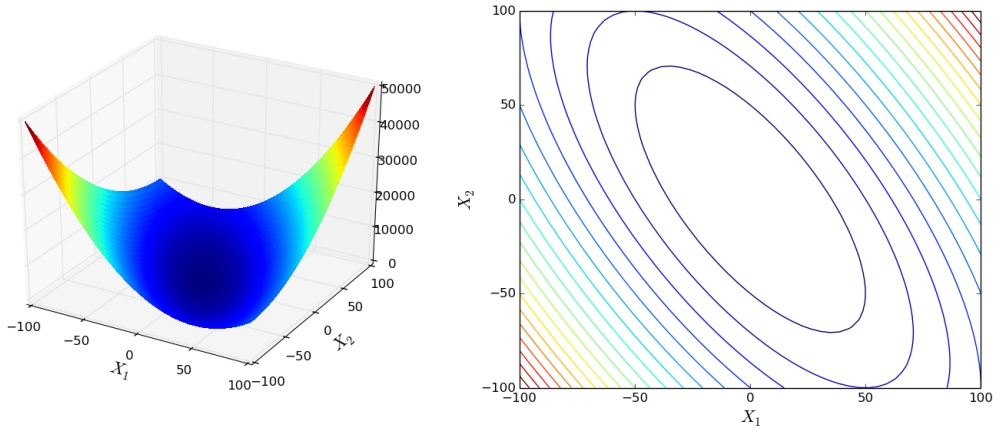


FIGURE 4.14 – Fonction Schwefel 12 : représentation et courbes de niveau

• **Schwefel 2.21**
L'expression analytique de la fonction Schwefel 2.21 est donnée par

$$f(x) = \max_i \{|x_i|, 1 \leq i \leq n\},$$

où n est le nombre de variables et $x \in [-100, 100]^n$. Elle possède un seul minimum en

$$x_i^\star = 0, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.15.

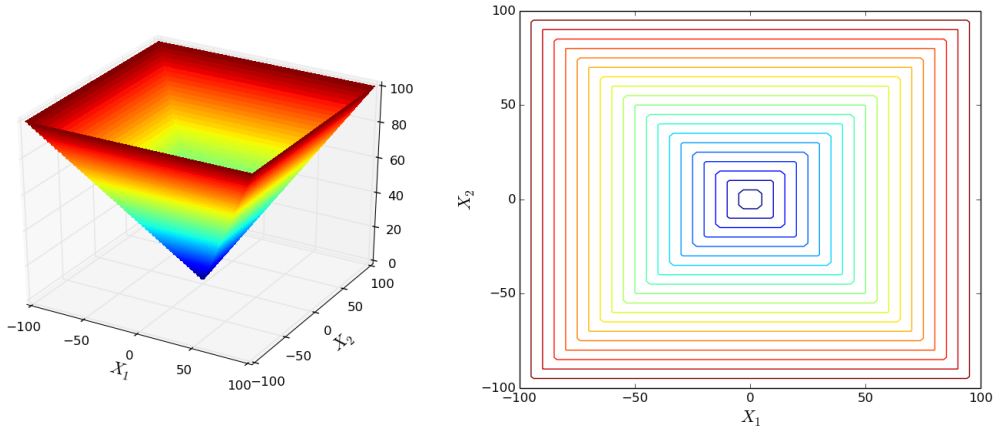


FIGURE 4.15 – Fonction Schwefel 2.21 : représentation et courbes de niveau

• **Schwefel 2.22**
L'expression analytique de la fonction Schwefel 2.22 est donnée par

$$f(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|,$$

où n est le nombre de variables et $x \in [-10, 10]^n$. Elle possède un seul minimum en

$$x_i^\star = 0, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.16.

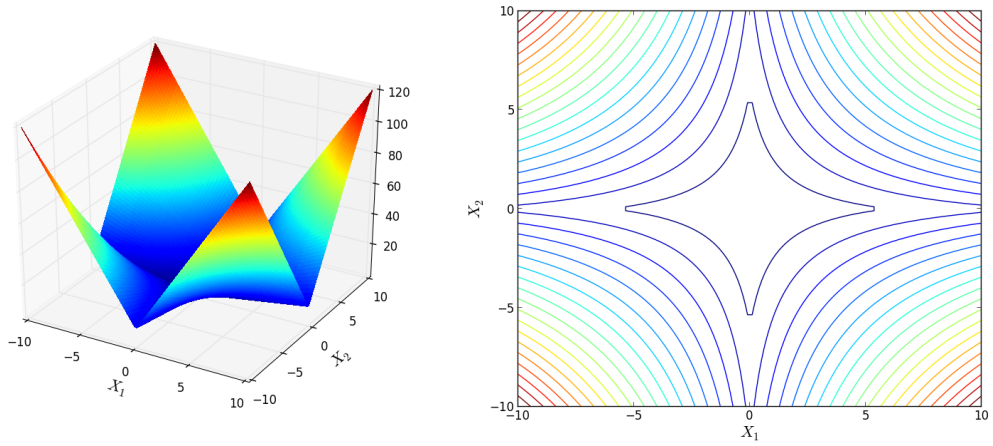


FIGURE 4.16 – Fonction Schwefel 2.22 : représentation et courbes de niveau

• **Schwefel 2.26**

L'expression analytique de la fonction Schwefel 2.26 est donnée par

$$f(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{|x_i|}),$$

où n est le nombre de variables et $x \in [-500, 500]^n$. Elle est multimodale, son minimum global se trouve en

$$x_i^\star = 420.9687, \qquad i = 1, \ldots, n$$

et $f(x^\star) = -418.8929n$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.17.

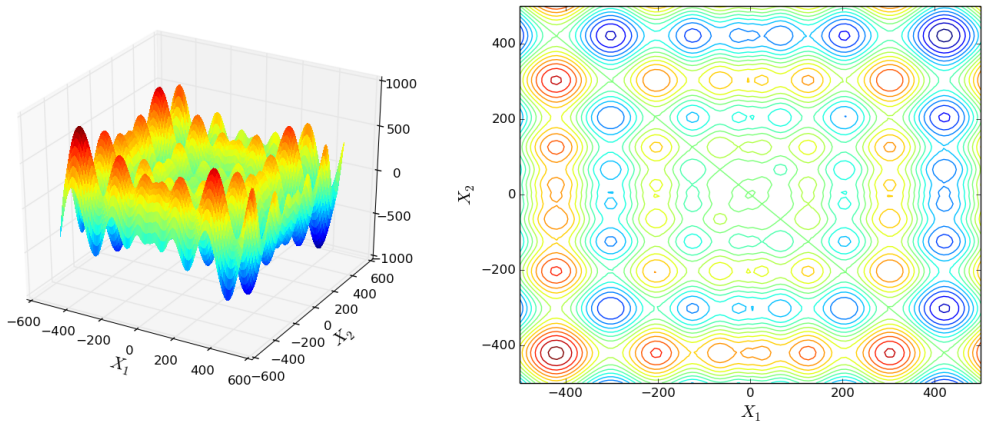


FIGURE 4.17 – Fonction Schwefel 2.26 : représentation et courbes de niveau

• **Sphère**
L'expression analytique de la fonction Sphère est donnée par

$$f(x) = \sum_{i=1}^{n} x_i^2,$$

où n est le nombre de variables et $x \in [-100, 100]^n$. Elle possède un seul minimum en

$$x_i^\star = 0, \qquad i = 1, \ldots, n$$

et $f(x^\star) = 0$. Une représentation de cette fonction pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.18.

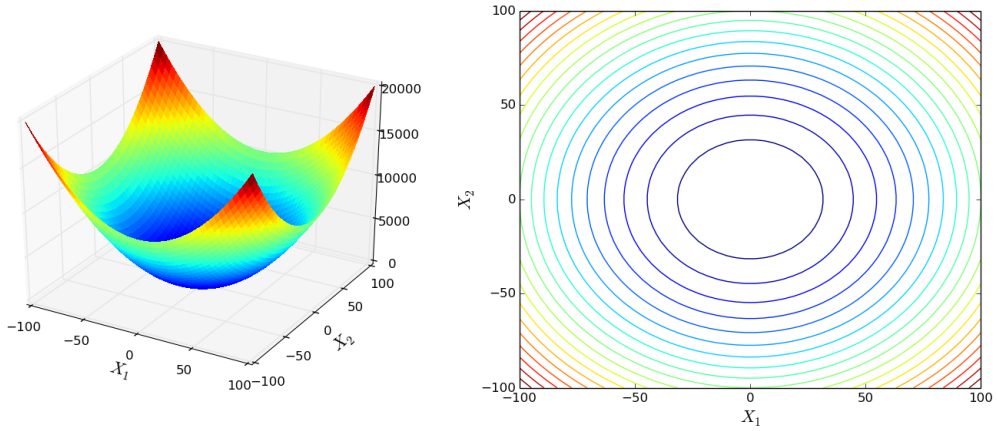


FIGURE 4.18 – Fonction Sphère : représentation et courbes de niveau

• **Step**
L'expression analytique de la fonction Step est donnée par

$$f(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2,$$

où n est le nombre de variables, $x \in [-100, 100]^n$ et $\lfloor a \rfloor$ désigne la partie entière de a. Elle possède un ensemble de mimima globaux qui est donné par l'ensemble des points qui vérifient

$$-0.5 \leq x_i^\star < 0.5, \qquad i = 1, \ldots, n.$$

La valeur de la fonction y est nulle. Une représentation de celle-ci pour $n = 2$ ainsi que ses courbes de niveau est fournie à la FIGURE 4.19. Un zoom autour de l'ensemble des optima globaux est également fourni à la FIGURE 4.20.

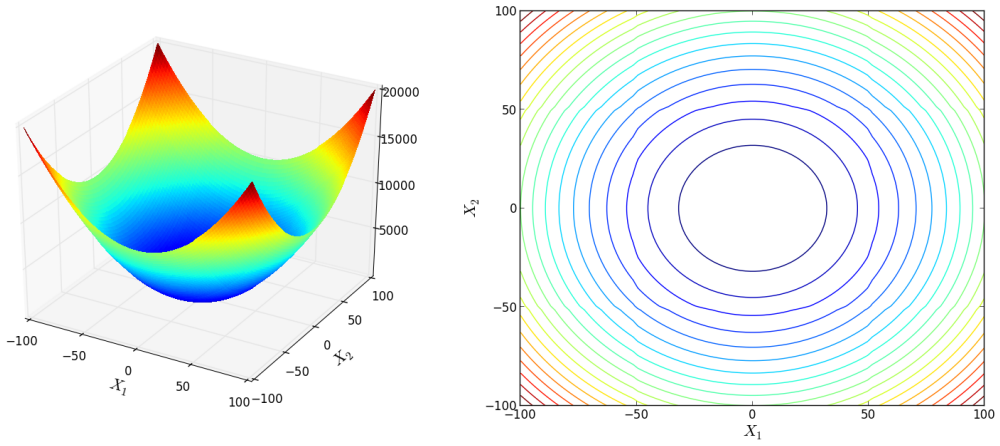


Figure 4.19 – Fonction Step : représentation et courbes de niveau

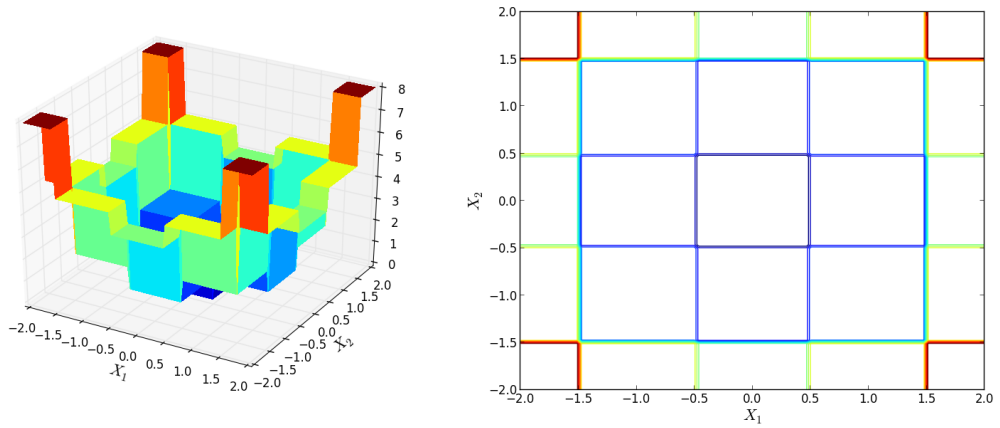


Figure 4.20 – Fonction Step : zoom autour de l'ensemble des optima globaux

4.2 Les fonctions transformées

Parmi les fonctions présentées précédemment et plus généralement, parmi les fonctions test couramment utilisées en optimisation, nombreuses sont celles qui sont symétriques. Cela n'est pas représentatif des fonctions rencontrées lors d'applications concrètes. Pour répondre à ce problème, Li, Tang, Omidvar, Yang et Qin [14] proposent d'appliquer, aux fonctions test couramment utilisées, une transformation non linéaire afin de casser la symétrie de ces fonctions [10]. Celle-ci est définie par

$$T_{asy}^{\beta} : \mathbb{R}^n \to \mathbb{R}^n, x_i \mapsto \begin{cases} x_i^{1+\beta\frac{i-1}{n-1}\sqrt{x_i}} & \text{si } x_i > 0, \\ x_i & \text{sinon} \end{cases} \qquad i = 1, \ldots, n.$$

Dans le cadre de notre travail, nous appliquons cette transformation aux fonctions Ackley, Elliptique, Rastrigin, Rosenbrock et Sphère en choisissant $\beta = 0.2$ (valeur couramment utilisée dans [14]). Les nouvelles fonctions obtenues sont respectivement nommées AckleyTr, ElliptiqueTr, RastriginTr, RosenbrockTr et SphèreTr.

Soit $f : D \subseteq \mathbb{R}^n \to \mathbb{R}, x \mapsto f(x)$, une fonction « classique », la fonction transformée f_{Tr} qui lui correspond est définie par

$$f_{Tr} : D \subseteq \mathbb{R}^n \to \mathbb{R}, x \mapsto f(T_{asy}^{0.2} x).$$

À la FIGURE 4.21, nous illustrons l'effet de cette transformation sur les courbes de niveau de la fonction Sphère à deux dimensions. De par la définition de T_{asy}^{β}, seuls les quadrants I, II et IV sont modifiés.

4.3 Les fonctions avec rotation

Une façon simple de construire des fonctions non séparables à partir de fonctions séparables est d'appliquer une rotation à l'ensemble des variables. Cette idée est utilisée par Li, Tang, Omidvar, Yang et Qin [14] pour construire leur ensemble de fonctions test. Dans le cadre de notre travail, nous appliquons une rotation aux fonctions Ackley, Elliptique et Rastrigin. Les nouvelles fonctions obtenues sont respectivement nommées AckleyRot, ElliptiqueRot et RastriginRot.

Soit $f : D \subseteq \mathbb{R}^n \to \mathbb{R}, x \mapsto f(x)$, une fonction « classique » et $M \in \mathbb{R}^n \times \mathbb{R}^n$ une matrice de rotation, la fonction avec rotation f_{Rot} qui lui correspond est définie par

$$f_{Rot} : D \subseteq \mathbb{R}^n \to \mathbb{R}, x \mapsto f(Mx).$$

À la FIGURE 4.22, nous illustrons l'effet d'une rotation sur les courbes de niveau de la fonction Elliptique à deux dimensions. La matrice de rotation $M \in \mathbb{R}^2 \times \mathbb{R}^2$ correspond à une rotation de 45° et est donnée par

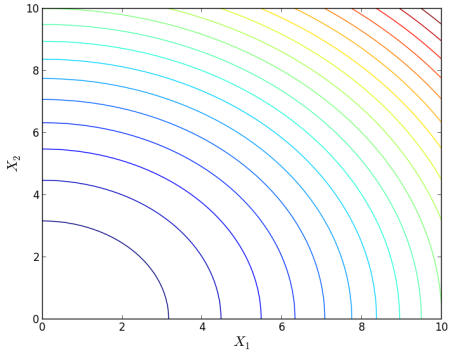
$$M = \begin{pmatrix} \cos 45^\circ & -\sin 45^\circ \\ \sin 45^\circ & \cos 45^\circ \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix}.$$

Dès lors, l'expression de la fonction Elliptique avec rotation à deux dimensions est donnée par

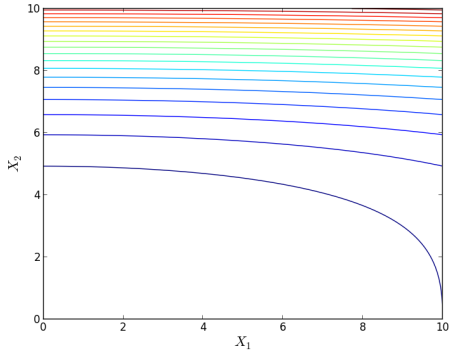
$$\begin{aligned} f : D \subseteq \mathbb{R}^n \to \mathbb{R}, x \mapsto f(x) &= \sum_{i=1}^{2} 10^{6(i-1)} y_i^2 = y_1^2 + 10^6 y_2^2 \text{ où } \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= \frac{1\ 000\ 001}{2}(x_1^2 + x_2^2) + 999\ 999 x_1 x_2. \end{aligned}$$

Cette fonction est non séparable. En effet, en suivant la définition 3.1, nous obtenons :

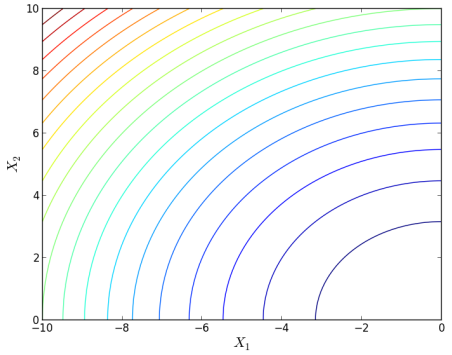
$$f(0,1) < f(0,2) \quad \not\Rightarrow \quad \forall y \in \mathbb{R} \quad f(y,1) < f(y,2) \quad \text{car} \quad f(-2,1) > f(-2,2).$$



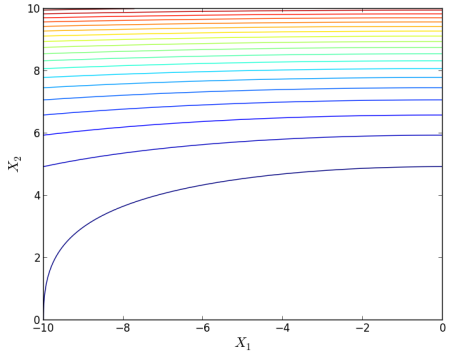
(a) Sphère : quadrant I



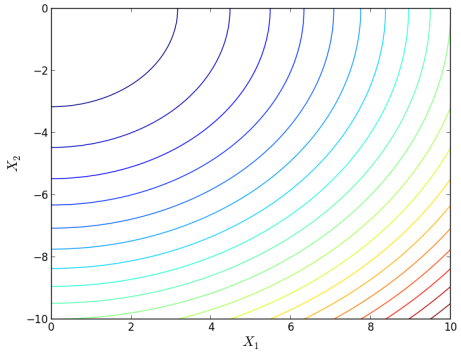
(b) SphèreTr : quadrant I



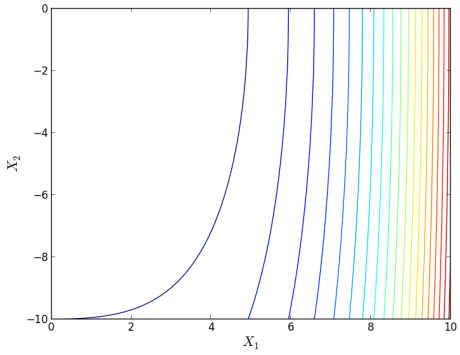
(c) Sphère : quadrant II



(d) SphèreTr : quadrant II

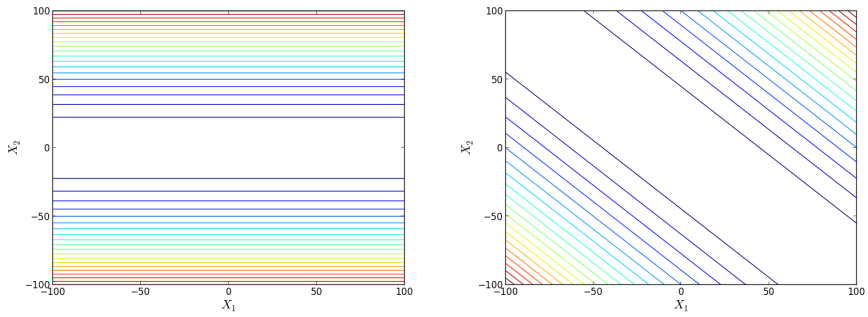


(e) Sphère : quadrant IV



(f) SphèreTr : quadrant IV

Figure 4.21 – Courbes de niveau des fonctions Sphère et SphèreTr



(a) Elliptique (b) EllipticqueRot

Figure 4.22 – Courbes de niveau des fonctions Elliptique et EllipticqueRot

4.4 Les graphes de convergence et les tableaux statistiques

Les différents résultats de ce mémoire sont présentés au moyen de graphes de convergence et de tableaux statistiques. Nous commençons donc par expliquer la manière dont ils sont conçus et les enseignements que l'on peut en tirer.

Tout d'abord, il est important de rappeler que les algorithmes évolutionnaires sont des algorithmes stochastiques. Cela signifie que deux exécutions successives de tels algorithmes peuvent donner des résultats différents. Dès lors, dans le but d'obtenir des résultats fiables, l'ensemble des tests que nous effectuons sont réalisés sur 100 différents runs. Autrement dit, nous lançons 100 fois le même algorithme et nous traitons les résultats obtenus par chacun des algorithmes grâce à différents outils.

Le premier d'entre eux est le graphe de convergence. Sur celui-ci sont représentées plusieurs courbes correspondant à une version d'algorithme. Chacune des courbes représente l'évolution de la moyenne, sur les 100 runs, de la valeur de la fonction objectif en fonction du nombre d'évaluations de cette fonction. De plus, pour une meilleure lisibilité, les valeurs de fonctions sont, dans la plupart des cas, représentées suivant une échelle logarithmique. Les graphes de convergence permettent de comparer les solutions moyennes obtenues par les différentes versions testées mais également de distinguer la vitesse de convergence de chacune d'entre elles.

Un deuxième outil d'analyse est un tableau reprenant les statistiques sur la valeur de la fonction objectif de la solution trouvée lors des 100 différents runs, les 5 valeurs les plus grandes étant exclues (pratique couramment utilisée au sein de l'équipe Minamo à Cenaero). Les statistiques sont les suivantes : la meilleure solution ainsi que la pire solution obtenue parmi les 95 runs restants, la moyenne, la valeur médiane et l'écart-type. Contrairement à la moyenne, la médiane possède l'avantage de ne pas être sensible aux valeurs extrêmes.

Enfin, un dernier outil de comparaison entre différentes versions est le temps d'exécution et la mémoire utilisée. Ces derniers sont également analysés dans ce mémoire.

Chapitre 5

Algorithme évolutionnaire distribué dans Minamo

À présent, nous nous intéressons plus particulièrement à l'algorithme évolutionnaire distribué basé sur un mécanisme de décompositions aléatoires présenté à la Section 3.1. Pour rappel, celui-ci consiste à séparer aléatoirement l'ensemble des variables de la fonction que l'on souhaite optimiser en sous-ensembles, formant ainsi plusieurs sous-populations qui évoluent chacune avec leur propre algorithme évolutionnaire. Ce processus est itéré pour un certain nombre de cycles fixé au préalable. Ce chapitre reprend la description de l'algorithme distribué via des décompositions aléatoires que nous implémentons dans MINAMO [1]. Celui-ci est dénommé `Minamo DistrEA` (pour *Minamo Distributed Evolutionary Algorithm*). Par la suite, nous testons ce nouvel algorithme et nous présentons les résultats. Pour ce faire, nous jouons sur les paramètres propres à l'algorithme distribué, à savoir le nombre de cycles et le nombre de sous-populations, afin de pouvoir les choisir au mieux. Enfin, nous comparons les performances de l'algorithme évolutionnaire distribué à celles de l'algorithme évolutionnaire classique de MINAMO (c'est-à-dire l'algorithme évolutionnaire actuel sans méta-modèles) sur des problèmes à 10, 50 et 500 variables.

Rappelons que la renommée de MINAMO repose sur sa stratégie d'algorithme évolutionnaire assisté par méta-modèles. Étant celle la plus utilisée à Cenaero, elle a bénéficié d'un investissement plus important ces dernières années, au contraire de l'algorithme évolutionnaire pur, sans méta-modèle. Une perspective de ce travail sera dès lors de développer un algorithme distribué utilisant cette stratégie assistée par méta-modèles.

5.1 L'implémentation

Cette section explicite, pas à pas, la description de l'algorithme `Minamo DistrEA`. Une description générale de celui-ci est proposée à l'Algorithme 5.1. Dans la suite des

1. Pour des questions de confidentialité, nous ne pouvons présenter ici-même les codes que nous avons implémentés.

Algorithme 5.1 : Minamo DistrEA

Étape 1 :

1. Construire une population pop de p individus.
2. Évaluer pop et trouver le meilleur individu $best$.
3. Poser $cycle = 0$.

Étape 2 :

1. Séparer pop en m sous-populations $pop_1, \ldots, pop_m$.
2. Évaluer $pop_1, \ldots, pop_m$ à l'aide de $best$.
3. Extraire l'individu « représentant » rep des sous-populations $pop_1, \ldots, pop_m$.
4. Évaluer rep.
5. Poser $generation = 0$.

Étape 3 :

1. Appliquer les opérateurs génétiques à $pop_1, \ldots, pop_m$.
2. Évaluer $pop_1, \ldots, pop_m$ à l'aide de rep.
3. Extraire le nouvel individu « représentant » rep des sous-populations $pop_1, \ldots, pop_m$.
4. Évaluer rep.
5. Si $fitness(rep) < fitness(best)$,
   $best = rep$.
6. Si $generation < maxGenerations$,
   - Incrémenter $generation$.
   - Aller à l'étape 3.1.

Étape 4 :

1. Si $cycle < maxCycles$,
   - Incrémenter $cycle$.
   - Construire une population pop de p individus contenant l'individu $best$.
   - Aller à l'étape 2.

   Sinon,
   - Fin de l'optimisation.
   - Solution = $fitness(best)$.

explications, nous supposons, sans perdre de généralité, que nous résolvons un problème d'optimisation de la forme suivante :

$$\min_{x \in D} f(x_1, \ldots, x_n) \qquad \text{où } f : D \subseteq \mathbb{R}^n \to \mathbb{R}.$$

La première étape est l'étape d'initialisation. Elle a pour but de construire une population *pop* de p individus. Un individu étant un point appartenant au domaine de définition D de la fonction f, la population *pop* est un ensemble de p points de D. Ces points sont générés de telle sorte que l'espace soit au mieux rempli grâce à la méthode *Latinized CVT* présentée à la Section 2.2.2. Par la suite, *pop* est évaluée, ce qui signifie que chacun de ses individus est évalué au moyen de la fonction objectif f. L'individu possédant le fitness (i.e., la valeur de la fonction objectif) le plus bas est noté *best* et la variable *cycle* est fixée à zéro.

La deuxième étape est accomplie au début de chaque nouveau cycle. Elle consiste principalement à diviser la population *pop* en sous-populations $pop_1, \ldots, pop_m$. Pour ce faire, nous commençons par diviser aléatoirement l'ensemble des indices des variables $I = \{1, \ldots, n\}$ en m sous-ensembles $I_1, \ldots, I_m$ contenant chacun s indices ($s = \frac{n}{m}$)[2]. Chaque sous-population pop_i ($i \in \{1, \ldots, m\}$) est alors composée des individus de *pop*, à l'exception que ceux-ci sont caractérisés par les variables correspondant aux indices de I_i uniquement. Nous illustrons ce mécanisme de division en sous-populations au moyen de l'Exemple 5.1.

Exemple 5.1
Soit une fonction

$$\begin{array}{rccc} f : & [-100, 100]^6 & \to & \mathbb{R} \\ & (x_1, x_2, x_3, x_4, x_5, x_6) & \mapsto & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 \end{array} .$$

Soit une population pop de 5 *individus que l'on souhaite diviser en* 3 *sous-populations. L'ensemble des indices des variables* $I = \{1, 2, 3, 4, 5, 6\}$ *est aléatoirement réparti en* 3 *sous-ensembles*

$$I_1 = \{1, 3\}, \qquad I_2 = \{4, 5\} \qquad et \qquad I_3 = \{2, 6\}.$$

La construction des sous-populations correspondant à ces sous-ensembles est illustrée à la FIGURE *5.1. Sur celle-ci, sont représentées différentes (sous-)populations. Dans chacune d'entre elles, une ligne représente un individu tandis que chaque colonne correspond à une variable (*$X01, X02, X03, X04, X05, X06$*) ou bien à la valeur de la fonction objectif (F).*

Chacune des sous-populations doit ensuite être évaluée à l'aide de l'individu *best*. En effet, chaque individu de la sous-population pop_i est caractérisé par s variables de I_i et, f étant une fonction à n variables, doit être complété par $n - s$ variables pour être évalué au moyen de f. Ces $n - s$ variables sont les variables de *best* n'appartenant pas à I_i. Ce procédé est éclairci à l'Exemple 5.2.

2. Si $\frac{n}{m} \notin \mathbb{N}$, l'ensemble $I = \{1, \ldots, n\}$ est divisé en m_1 sous-ensembles contenant s indices et m_2 sous-ensembles contenant $s + 1$ indices de telle sorte que $m_1 + m_2 = m$ et $m_1 \times s + m_2 \times (s + 1) = n$.

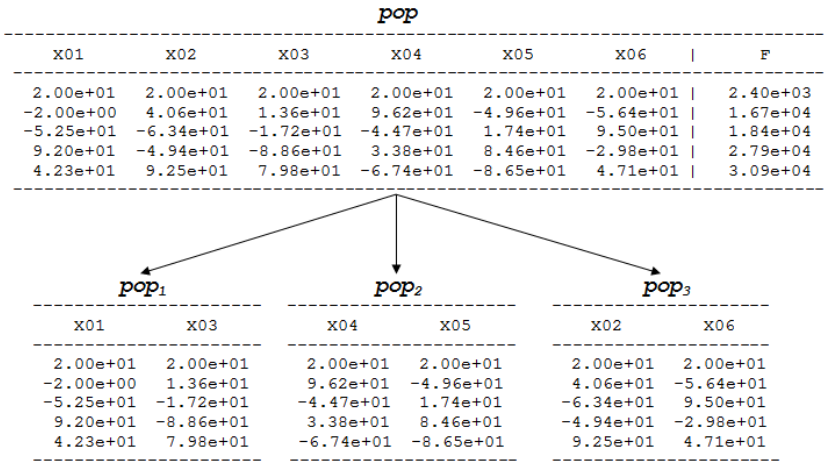


FIGURE 5.1 – Illustration de la division en sous-populations

Exemple 5.2

Cet exemple constitue la suite de l'Exemple 5.1 et illustre, à la FIGURE *5.2, l'évaluation des sous-populations. Nous pouvons y constater que les individus de la sous-population* pop_1 *sont évalués à partir des variables qui la caractérise,* $X01$ *et* $X03$*, ainsi que des variables de best n'appartenant pas à* $I_1 = \{1, 3\}$*, à savoir* $X02, X04, X05$ *et* $X06$*.*

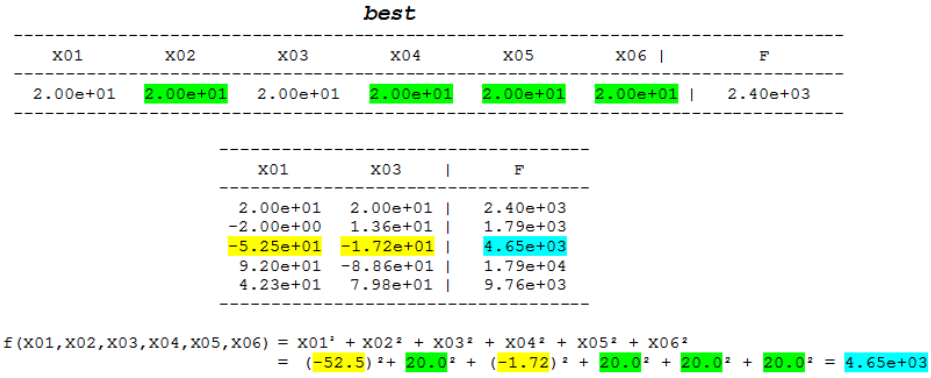


FIGURE 5.2 – Illustration de l'évaluation des sous-populations

Les sous-populations étant évaluées, il faut en extraire l'individu « représentant » *rep* des sous-populations. Celui-ci est construit en concaténant les représentants tels que présentés à la Section 3.2 de chacune des sous-populations. Dans notre cas, le représentant de chaque sous-population est le meilleur individu de celle-ci, c'est-à-dire celui possédant le fitness le plus bas. Ce procédé est illustré à l'Exemple 5.3.

Exemple 5.3

Cet exemple est la suite de l'Exemple 5.2 et est illustré par la FIGURE *5.3. Nous pouvons y voir que l'individu rep est construit à partir des variables* $X01$ *et* $X03$*,* $X04$ *et* $X05$ *ainsi que* $X02$ *et* $X06$ *des meilleurs individus respectifs des sous-populations* pop_1*,* pop_2 *et* pop_3*.*

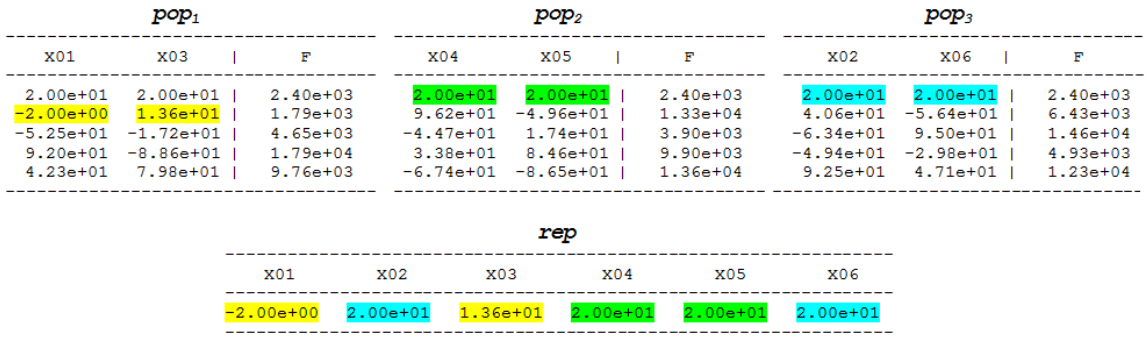


FIGURE 5.3 – Illustration de l'extraction du représentant

Pour conclure l'étape 2, nous évaluons l'individu *rep* et nous fixons la variable *generation* à zéro.

La troisième étape comprend les différentes actions effectuées lors de chaque génération. Nous commençons par appliquer les opérateurs génétiques, c'est-à-dire les étapes de *Sélection, Reproduction/Mutation, Population suivante* décrites à la Section 1.4 à chacune des sous-populations. Ces étapes, propres à MINAMO, sont à caractère confidentiel et ne peuvent donc être explicitées davantage ici-même. Chacune des sous-populations est ensuite évaluée à l'aide de l'individu *rep* (de la même façon que la population est évaluée à partir de l'individu *best* à la FIGURE 5.2). Dès lors, un nouveau représentant peut être extrait des sous-populations (comme illustré à la FIGURE 5.3). Celui-ci constitue à présent le nouvel individu *rep*. Il est évalué et si son fitness est meilleur que celui de l'individu *best*, ce dernier est remplacé par *rep*. Finalement, si le nombre maximum de générations n'est pas atteint, la variable *generation* est incrémentée et la troisième étape est de nouveau appliquée, sinon nous passons à la quatrième étape.

La quatrième étape est opérée à la fin de chaque cycle. Si le nombre maximum de cycles n'est pas atteint, elle consiste à incrémenter la variable *cycle*, à construire une population de p individus contenant l'individu *best* et à passer à l'étape 2. La population construite est, de nouveau, obtenue au moyen de la méthode *LCVT*. Sinon, l'optimisation est terminée et la solution est donnée par le fitness de l'individu *best*.

Finalement, nous concluons cette section en analysant le nombre d'évaluations de fonctions nécessaires lors d'une exécution de l'algorithme en fonction des paramètres choisis. La fonction objectif est évaluée p fois lors de l'étape 1 pour évaluer *pop*. Lors de l'étape 2, elle est évaluée $p \times m$ fois afin d'évaluer les m sous-populations et une fois pour évaluer l'individu « représentant » *rep*. Il en est de même lors de chaque boucle de l'étape 3. Quant à l'étape 4, elle ne requiert pas d'évaluations. Pour résumer, le nombre d'évaluations de fonctions (FE) est donné par

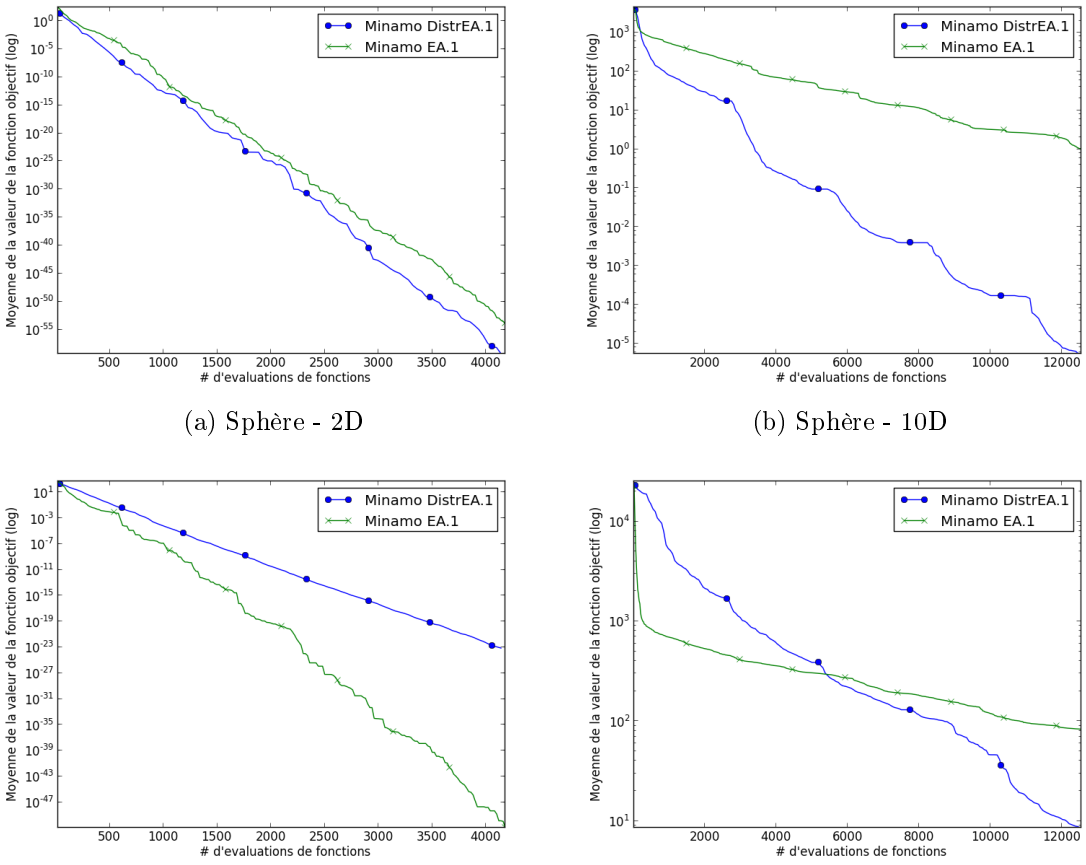
$$FE = p + [(p \times m + 1) \times (maxGenerations + 1)] \times maxCycles.$$

Par la suite, nous approximerons celui-ci de la façon suivante :

$$FE \approx p \times m \times maxGenerations \times maxCycles. \tag{5.1}$$

5.2 La présentation des résultats

Nous allons à présent valider notre algorithme distribué `Minamo DistrEA` et présenter nos résultats. La présentation de ceux-ci est réalisée au moyen de graphes de convergence (moyenne sur 100 runs) et de tableaux statistiques (voir Section 4.4). De manière générale, l'algorithme `Minamo DistrEA` ne fournit pas de meilleures performances que l'algorithme évolutionnaire classique, dénommé `Minamo EA`, pour des fonctions de faible dimension (2 ou 3 variables) mais le surclasse lorsque la dimension devient relativement grande (plus de 10 variables). Ce phénomène est illustré à la FIGURE 5.4 sur base des paramètres décrits à la TABLE 5.1. On peut y voir que pour la fonction Sphère - 2D, les deux algorithmes fournissent des performances similaires car celle-ci est de caractère séparable alors que pour la fonction Schwefel 12 - 2D, l'algorithme `Minamo EA` fournit de meilleurs résultats car la fonction est de caractère non séparable. Par contre, pour les fonctions Sphère - 10D et Schwefel 12 - 10D, l'algorithme `Minamo DistrEA` est meilleur que l'algorithme `Minamo EA`, quel que soit le type de fonction.



(a) Sphère - 2D

(b) Sphère - 10D

(c) Schwefel 12 - 2D

(d) Schwefel 12 - 10D

FIGURE 5.4 – Graphes de convergence des algorithmes décrits à la TABLE 5.1

Minamo	Cycles	Générations	Taille de la population	Sous-populations
EA - 2D	-	208	20	-
EA - 10D	-	626	20	-
DistrEA - 2D	1	100	20	2
DistrEA - 10D	5	40	20	3

Table 5.1 – Description des algorithmes pour comparer différentes dimensions

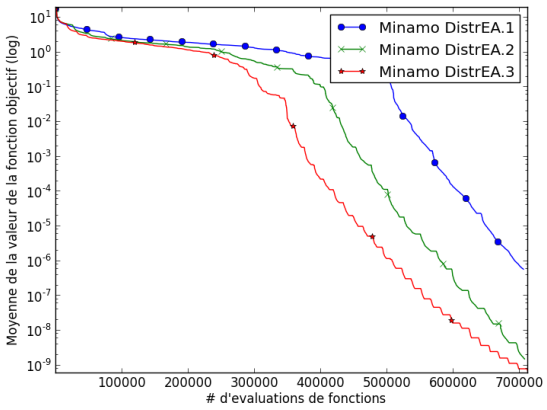
Dans la suite de cette section, nous jouons sur les paramètres propres à l'algorithme distribué, à savoir le nombre de cycles et le nombre de sous-populations, afin de pouvoir les choisir au mieux. Nous comparons ensuite les performances de l'algorithme évolutionnaire distribué à celles de l'algorithme évolutionnaire classique sur des fonctions à 50 et 500 variables.

5.2.1 Influence du nombre de cycles

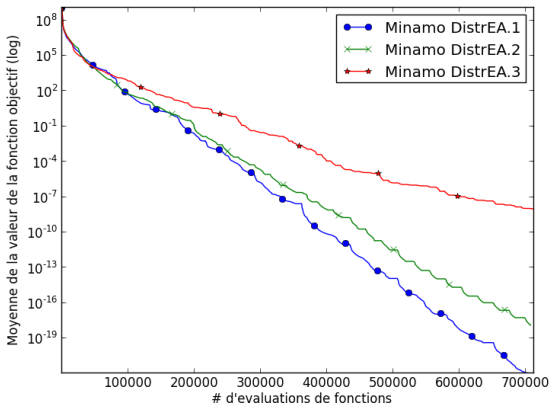
Dans cette section, nous testons l'influence du nombre de cycles sur les performances de l'algorithme `Minamo DistrEA` sur les différentes fonctions test. Pour rappel, à chaque cycle, les différentes sous-populations, qui sont caractérisées par les variables groupées aléatoirement en début de cycle, sont optimisées grâce à l'algorithme classique de Minamo ; la coopération entre chacune d'entre elles ayant lieu lors de l'évaluation du fitness. La question à laquelle nous tentons de répondre dans cette section est de savoir s'il est préférable, selon les fonctions optimisées, de choisir un petit ou un grand nombre de cycles. A priori, l'avantage de réaliser un plus grand nombre de cycles est d'effectuer un plus grand nombre de groupements aléatoires et ainsi d'avoir plus de chances de grouper des variables qui interagissent entre elles lors d'au moins un cycle. Cependant, l'inconvénient est de repartir, lors de chaque nouveau cycle, d'une nouvelle population (pour rappel, seul le meilleur individu est transmis de cycle en cycle). Le processus de convergence de la population vers la solution reprend donc presque à zéro.

Nous choisissons d'effectuer nos tests sur l'ensemble des fonctions présentées au chapitre précédent en fixant le nombre de variables à 50. Trois versions de notre algorithme distribué sont comparées. Leur nom ainsi que la description de leurs paramètres sont repris à la Table 5.2. Conformément à l'équation (5.1) et dans le but de comparer les différentes versions pour un même nombre d'évaluations de fonctions, nous diminuons le nombre de générations lorsque nous augmentons le nombre de cycles.

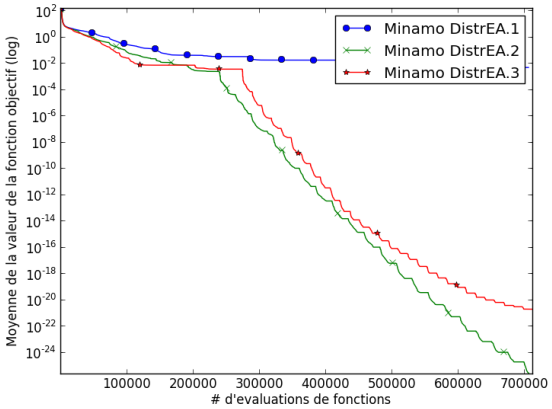
Une première analyse consiste à comparer les graphes de convergence des différentes versions sur les fonctions test. Les graphes de dix d'entre elles sont repris à la Figure 5.5 et à la Figure 5.6.



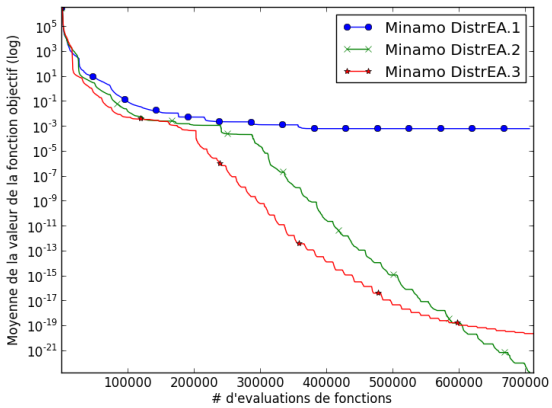
(a) Ackley - 50D



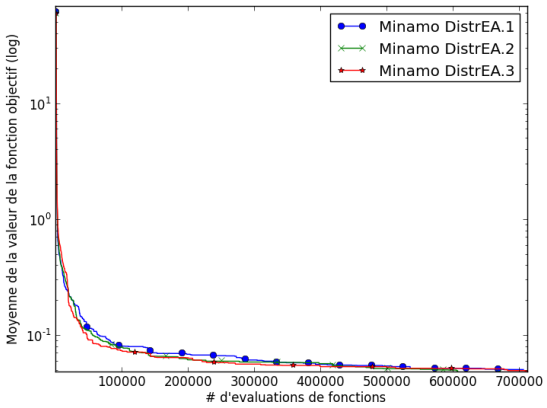
(b) Elliptique - 50D



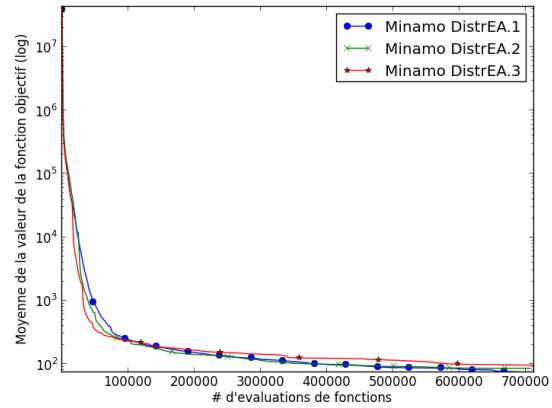
(c) Generalized Penalized 1 - 50D



(d) Generalized Penalized 2 - 50D

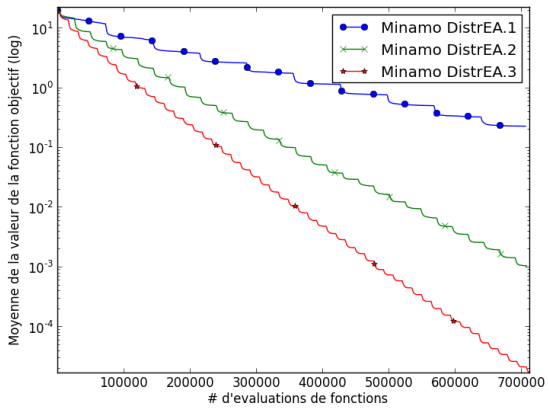


(e) Quartique - 50D

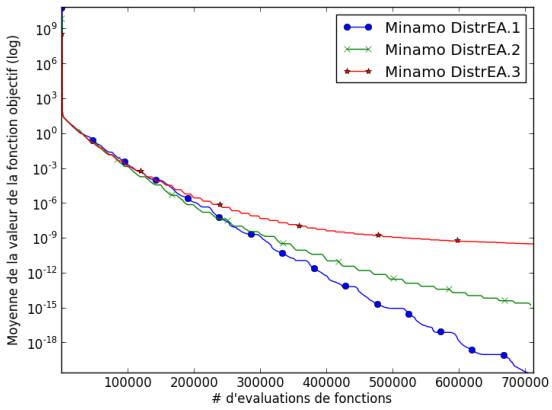


(f) Rosenbrock - 50D

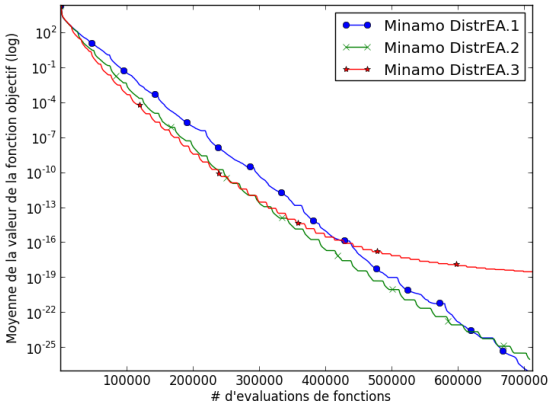
Figure 5.5 – Graphes de convergence des algorithmes décrits à la Table 5.2



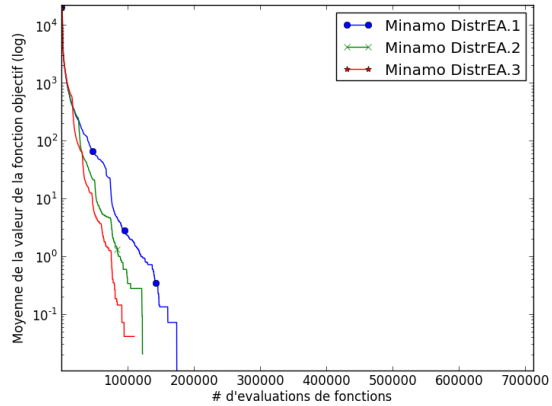
(a) Schwefel 2.21 - 50D



(b) Schwefel 2.22 - 50D



(c) Sphère - 50D



(d) Step - 50D

FIGURE 5.6 – Graphes de convergence des algorithmes décrits à la TABLE 5.2

MINAMO	Cycles	Générations	Taille de la population	Sous-populations
DistrEA.1	10	400	35	5
DistrEA.2	30	133	35	5
DistrEA.3	50	80	35	5

TABLE 5.2 – Description des algorithmes pour tester l'influence du nombre de cycles

Nous pouvons voir, sur ces figures, que le constat n'est pas univoque. Selon les cas, il est préférable de faire plus ou moins de cycles.

- Pour les fonctions Elliptique, Schwefel 2.22 et Sphère, les résultats obtenus par les différents algorithmes sont d'autant meilleurs que le nombre de cycles est petit.

Cela n'est pas surprenant. Ces fonctions étant de caractère séparable, le nouveau groupement aléatoire lors de chaque cycle n'apporte aucun avantage et le fait de repartir d'une nouvelle population lors de chaque cycle ralentit considérablement le processus de convergence de la population vers la solution.

- Pour les fonctions Ackley, Generalized Penalized 1 et Generalized Penalized 2, l'algorithme `Minamo DistrEA.1`, qui n'effectue que 10 cycles, est toujours surclassé par les deux autres algorithmes. Ces fonctions possédant des propriétés de non-séparabilité, ont donc beaucoup de variables qui interagissent entre elles (voir, par exemple, Figure 4.4). Si le nombre de cycles est trop faible, il n'y a pas assez de groupements aléatoires et de nombreuses variables qui interagissent entre elles ne sont que très rarement, voire jamais, groupées dans la même sous-population. Dès lors, il est relativement difficile de trouver l'optimum global. Cependant, lorsque nous comparons les algorithmes `Minamo DistrEA.2` (30 cycles) et `Minamo DistrEA.3` (50 cycles), `Minamo DistrEA.2` distance `Minamo DistrEA.3` en début d'optimisation mais est rejoint voire dépassé par ce dernier en fin d'optimisation. Cela est dû au fait qu'en début d'optimisation, un grand nombre de cycles est nécessaire pour pouvoir gérer le caractère non séparable de ces fonctions. Néanmoins, dans un voisinage suffisamment proche de l'optimum global, celles-ci sont de caractère séparable (voir, par exemple, Figure 4.5) et un plus petit nombre de cycles permet alors de mieux converger vers la solution.
- Pour la fonction Schwefel 2.21, les résultats obtenus par les différents algorithmes sont d'autant meilleurs que le nombre de cycles est grand. Cela est dû au fait que la propriété de non-séparabilité de cette fonction est aussi bien vérifiée globalement que localement, dans un voisinage proche de l'optimum (voir, notamment, Figure 4.15). Dès lors, les nouveaux groupements aléatoires apportent un avantage considérable tout au long de l'optimisation.
- Quant aux fonctions Quartique, Rosenbrock et Step, les différents algorithmes fournissent sur celles-ci des performances relativement proches. Pour les fonctions Quartique et Rosenbrock, aucun d'entre eux ne converge relativement près de la solution tandis que pour la fonction Step, ils convergent tous les trois très rapidement vers la solution.

Il est également intéressant de comparer les différents algorithmes sur base du temps d'exécution et de la mémoire utilisée. En moyenne, pour l'ensemble des fonctions testées, le temps d'exécution pour l'algorithme `Minamo DistrEA.1` (10 cycles) est de 200 secondes alors qu'il est de 400 secondes pour `Minamo DistrEA.2` (30 cycles) et de 600 secondes pour `Minamo DistrEA.3` (50 cycles). Cela s'explique de la façon suivante : la génération d'une nouvelle population (DoE) lors de chaque nouveau cycle consomme la majeure partie du temps d'exécution. Dès lors, le temps d'exécution augmente proportionnellement avec le nombre de cycles. Quant à l'utilisation de la mémoire, le nombre de cycles ne semble pas l'influencer. En moyenne, pour l'ensemble des fonctions test et pour les différents algorithmes, la mémoire utilisée est de 240 Mb.

5.2.2 Influence du nombre de sous-populations

Dans cette section, nous testons l'influence du nombre de sous-populations sur les performances de l'algorithme `Minamo DistrEA` sur les différentes fonctions test. Pour rappel, lorsque nous optimisons une fonction à n variables et que nous divisons la population en m sous-populations, les individus de chacune d'entre elles sont caractérisés par $s = \frac{n}{m}$ variables. Cela signifie que plus le nombre de sous-populations choisi est grand, plus le nombre de variables par sous-population est petit. Il y a différents avantages à choisir un petit ou un grand nombre de sous-populations. L'avantage de travailler avec un grand nombre de sous-populations est donc de travailler sur des populations possédant peu de variables, cas dans lequel l'algorithme classique de MINAMO est relativement performant. Toutefois, si le nombre de sous-populations est trop grand, plusieurs variables interagissant les unes avec les autres ont très peu de chances d'être groupées ensemble pour au moins un cycle et les échanges de représentants lors de l'évaluation du fitness ne suffisent pas pour prendre en compte ces interactions. À l'inverse, si le nombre de sous-populations est petit, les chances de grouper des variables interagissant entre elles pour au moins un cycle sont relativement grandes mais l'algorithme classique de MINAMO est alors appliqué sur des populations possédant un grand nombre de variables, cas dans lequel celui-ci est moins performant.

De nouveau, nous effectuons nos tests sur l'ensemble des fonctions présentées au chapitre précédent en fixant le nombre de variables égal à 50. Trois versions de notre algorithme distribué sont comparées. Leur nom ainsi que la description de leurs paramètres sont repris à la TABLE 5.3. Comme précédemment, conformément à l'équation (5.1) et dans le but de comparer les différentes versions pour un même nombre d'évaluations de fonctions, nous diminuons le nombre de générations lorsque nous augmentons le nombre de sous-populations.

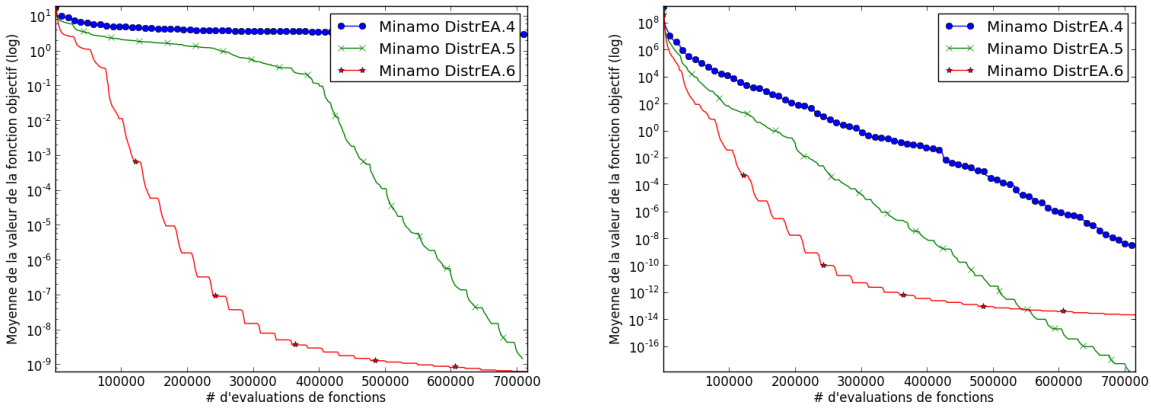
MINAMO	`Cycles`	`Générations`	`Taille de la population`	`Sous-populations`
`DistrEA.4`	30	333	35	2
`DistrEA.5`	30	133	35	5
`DistrEA.6`	30	67	35	10

TABLE 5.3 – Description des algorithmes pour tester l'influence du nombre de sous-populations

Une première analyse consiste à comparer les graphes de convergence des différentes versions sur les fonctions test. Les graphes de dix d'entre elles sont repris aux FIGURES 5.7, 5.8 et 5.9. Sur la plupart d'entre eux, nous pouvons voir qu'en début d'optimisation, plus le nombre de sous-populations est grand, plus les algorithmes sont performants. Ainsi, `Minamo DistrEA.6` (10 sous-populations) surclasse `Minamo DistrEA.5` (5 sous-populations) qui lui-même surclasse `Minamo DistrEA.4` (2 sous-populations). Un grand nombre de sous-populations aide donc à trouver une bonne région rapidement. Une fois cette bonne région trouvée, nous constatons que, dans de nombreux cas, `Minamo`

`DistrEA.5` rejoint voire dépasse `Minamo DistrEA.6`. En effet, il est préférable, en fin d'optimisation, d'avoir un petit nombre de sous-populations de telle sorte que chacune d'entre elles contienne une grande proportion de l'information globale. Quant à `Minamo DistrEA.4`, il ne parvient que très rarement à trouver cette bonne région. Le seul cas où il surclasse largement les deux autres algorithmes concerne la fonction Schwefel 12. L'expression analytique de cette fonction est caractérisée par les produits croisés entre chacune de ses variables, ce qui lui confère des propriétés de forte non-séparabilité. Dans ce cas, il est plus intéressant de travailler avec peu de sous-populations de façon à grouper de nombreuses variables entre elles.

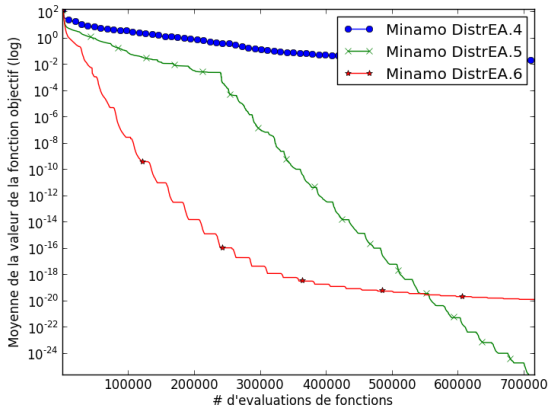
Nous comparons également les différents algorithmes sur base du temps d'exécution et de la mémoire utilisée. En moyenne, pour l'ensemble des fonctions testées, le temps d'exécution pour l'algorithme `Minamo DistrEA.4` (2 sous-populations) est de 460 secondes alors qu'il est de 430 secondes pour `Minamo DistrEA.5` (5 sous-populations) et de 410 secondes pour `Minamo DistrEA.6` (10 sous-populations). Cette différence s'explique de la façon suivante : chacun de ces algorithmes applique les opérateurs génétiques le même nombre de fois. Toutefois, lorsque le nombre de sous-populations est plus petit, les individus sont caractérisés par plus de variables et l'application des opérateurs génétiques nécessite donc plus de temps. Dès lors, plus le nombre de sous-populations est petit, plus le temps d'exécution est grand. Quant à l'utilisation de la mémoire, le nombre de sous-populations ne semble pas l'influencer. En moyenne, pour l'ensemble des fonctions test et pour les différents algorithmes, la mémoire utilisée est de 240 Mb.



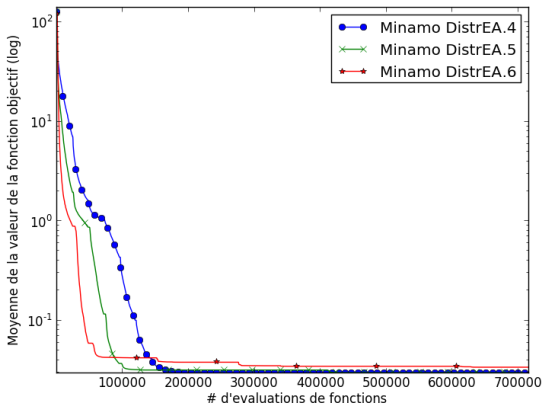
(a) Ackley - 50D

(b) Elliptique - 50D

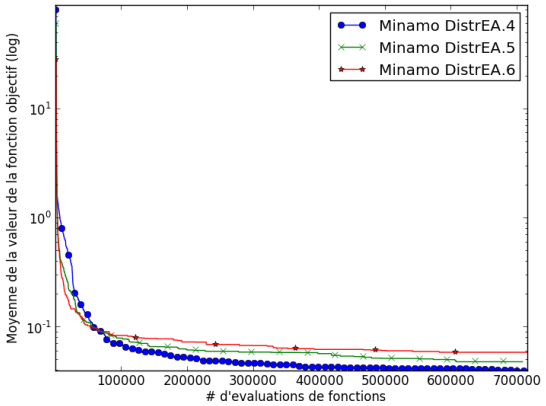
Figure 5.7 – Graphes de convergence des algorithmes décrits à la Table 5.3



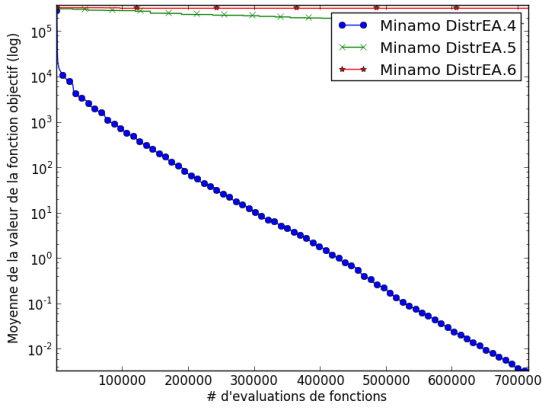
(a) Generalized Penalized 1 - 50D



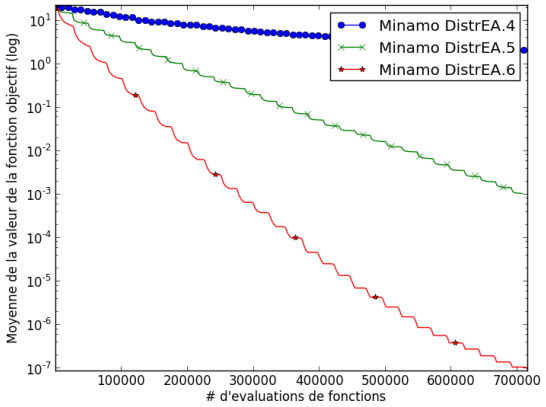
(b) Griewank - 50D



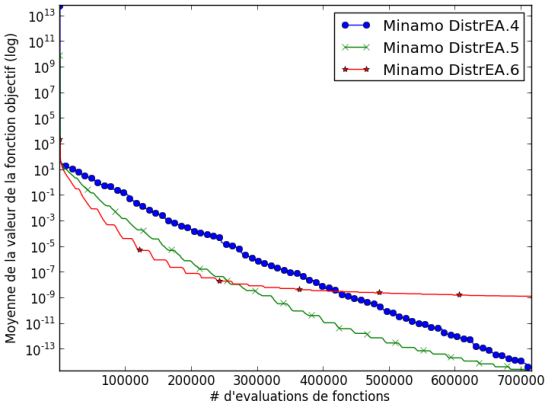
(c) Quartique - 50D



(d) Schwefel 12 - 50D

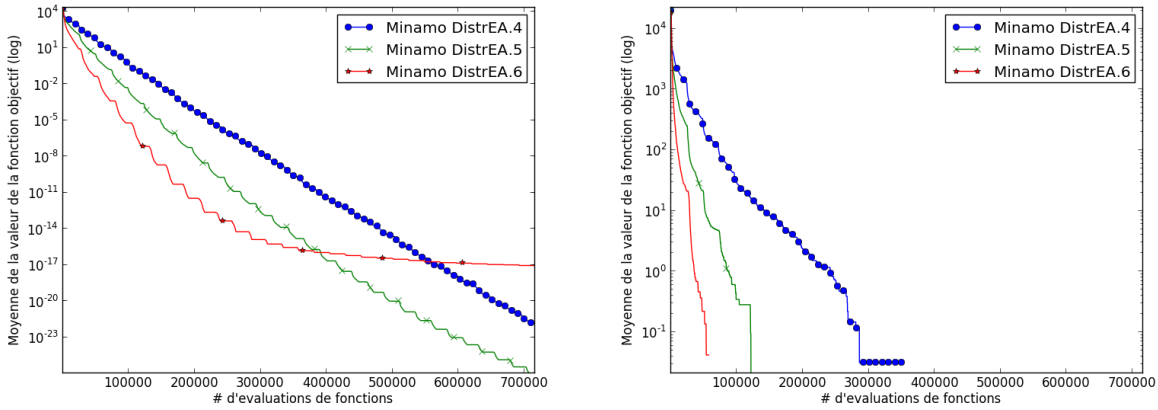


(e) Schwefel 2.21 - 50D



(f) Schwefel 2.22 - 50D

Figure 5.8 – Graphes de convergence des algorithmes décrits à la Table 5.3



(a) Sphère - 50D

(b) Step - 50D

FIGURE 5.9 – Graphes de convergence des algorithmes décrits à la TABLE 5.3

5.2.3 Comparaison avec l'algorithme évolutionnaire actuel de Minamo

À présent, nous testons l'algorithme `Minamo DistrEA` sur l'ensemble des fonctions test décrites au Chapitre 4 pour 50 et 500 variables. Les performances de ce dernier sont comparées à celle de l'algorithme évolutionnaire classique de MINAMO (c'est-à-dire l'algorithme évolutionnaire actuel sans méta-modèles). Celui-ci est dénommé `Minamo EA`. Dans le but de comparer ce qui est comparable, ces deux algorithmes sont comparés sur le même nombre d'évaluations de fonctions. Il est fixé à 7×10^5 pour les fonctions à 50 variables et à 9×10^6 pour les fonctions à 500 variables. De plus, pour chacun des algorithmes, la taille de la population est fixée à 35 individus en dimension 50 et à 100 individus en dimension 500. Pour l'algorithme `Minamo EA`, le nombre de générations est de 7 000 pour les fonctions de dimension 50 et de 90 000 pour les fonctions de dimension 500. Quant aux paramètres de l'algorithme `Minamo DistrEA`, ils varient selon les fonctions à optimiser et sont choisis en adéquation avec les caractéristiques de celles-ci. Ils sont repris à la TABLE 5.4. Dans ce dernier, *cyc*, *gen* et *sp* désignent, respectivement, le nombre de cycles, de générations et de sous-populations.

Pour rappel, comme expliqué à la Section 4.4, l'ensemble des tests que nous effectuons sont réalisés sur 100 différents runs. À la TABLE 5.5, est proposée, pour chacune des fonctions, la solution moyenne obtenue lors des 100 runs. Les graphes de convergence comparant les algorithmes `Minamo EA` et `Minamo DistrEA` pour les fonctions à 500 variables sont également fournis aux FIGURES 5.10 à 5.13. L'interprétation de ces résultats est on ne peut plus claire. L'algorithme `Minamo DistrEA` est significativement meilleur que l'algorithme `Minamo EA`. Sur la plupart des graphes de convergence, nous pouvons voir que l'algorithme `Minamo DistrEA` offre une amélioration de la fonction objectif bien plus grande que l'algorithme `Minamo EA` dès le début de l'optimisation. Cette dif-

	DistrEA-50D			DistrEA-500D		
	cyc	*gen*	*sp*	*cyc*	*gen*	*sp*
Ackley	50	80	5	50	36	50
Elliptique	10	400	5	30	150	20
GenPen1	30	133	5	50	90	20
GenPen2	30	133	5	50	90	20
Griewank	20	200	5	50	90	20
Quartique	30	333	2	50	360	5
Rastrigin	30	66	10	10	90	100
Rosenbrock	20	200	5	50	36	50
Schwefel 12	30	333	2	50	900	2
Schwefel 2.21	30	66	10	50	90	20
Schwefel 2.22	10	400	5	30	300	10
Schwefel 2.26	30	66	10	30	30	100
Sphère	20	200	5	30	300	10
Step	50	80	5	50	90	20
AckleyTr	30	66	10	30	30	100
ElliptiqueTr	30	66	10	30	30	100
RastriginTr	30	66	10	30	30	100
RosenbrockTr	50	80	5	50	90	20
SphèreTr	20	200	5	30	300	10
AckleyRot	30	66	10	30	30	100
ElliptiqueRot	30	333	2	50	180	10
RastriginRot	30	333	2	50	360	5

Table 5.4 – Paramètres de l'algorithme `Minamo DistrEA`

férence ne cesse de s'accroître par la suite. Au final, la solution obtenue par l'algorithme distribué est non seulement meilleure, mais de plus, il y a une grande différence d'ordre de grandeur entre les deux solutions. L'algorithme évolutionnaire classique ne permet pas de trouver une solution relativement précise pour des fonctions de dimension 50 et 500. Il est sujet à la *Malédiction de la dimension*[3]. Au contraire, l'algorithme distribué exploite pleinement l'avantage de travailler sur des sous-problèmes de dimension moindre et parvient ainsi à trouver une solution relativement précise pour la plupart des fonctions, qu'elles soient de caractère séparable, à l'instar des fonctions Sphère et Elliptique, ou non, comme les fonctions Schwefel 2.21, Generalized Penalized 1 et 2 ou encore Griewank.

Quelques fonctions font toutefois exception. D'une part, il y a les fonctions ElliptiqueTr et RastriginTr. Les résultats, très peu précis, obtenus pour celles-ci peuvent s'expliquer de la façon suivante. Les transformations appliquées dans le but de casser les symétries compliquent sensiblement la tâche de l'algorithme évolutionnaire appliqué à chacune des sous-populations. Dès lors, la résolution des différents sous-problèmes n'étant pas de qualité optimale, le problème global ne peut donc, lui non plus, être résolu efficacement. D'autre part, les fonctions Rosenbrock, Schwefel 12, AckleyRot, ElliptiqueRot et RastriginRot font également exception. Les fonctions Rosenbrock et Schwefel 12, de part leur expression analytique, et les fonctions AckleyRot, ElliptiqueRot et RastriginRot, de part la rotation appliquée à l'espace des variables, sont des fonctions à caractère non séparable pour lesquelles toutes les variables sont fortement

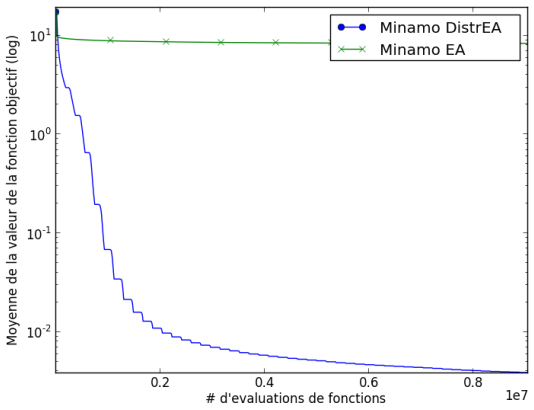
3. Terme introduit par Richard Bellman dans [2].

	50D-EA	50D-DistrEA	500D-EA	500D-DistrEA
Ackley	8.2892174e+00	**3.8659919e-10**	8.3549836e+00	**3.8251430e-03**
Elliptique	6.7459396e+04	**1.0685601e-22**	9.0304496e+05	**1.1389644e-05**
GenPen1	3.7802000e+01	**2.3746101e-26**	7.8872095e+00	**3.0736311e-09**
GenPen2	8.9643024e+00	**1.6373381e-23**	4.6067287e+02	**1.7608379e-12**
Griewank	3.4490642e-01	**1.2874647e-02**	2.3397531e+00	**1.1422252e-03**
Quartique	4.1510990e-02	**3.7719892e-02**	7.4560966e-01	**2.9938697e-01**
Rastrigin	4.8805579e+01	**1.4348359e+00**	4.1168691e+02	**1.6927300e+00**
Rosenbrock	1.2482982e+02	**4.4657126e+01**	4.2719612e+04	**7.1871933e+02**
Schwefel 12	1.1500687e+02	**1.1284238e-03**	1.4746046e+04	**8.7209365e+02**
Schwefel 2.21	2.0000000e+01	**8.5391820e-08**	1.9695255e+01	**4.7193498e-03**
Schwefel 2.22	7.6739473e-02	**2.4876855e-21**	3.3710664e+00	**1.1583291e-09**
Schwefel 2.26	-1.3674101e+04	**-1.9404062e+04**	-8.2877394e+04	**-1.9225367e+05**
Sphère	2.0962075e-03	**6.9512707e-28**	2.0209842e+02	**3.9208333e-19**
Step	7.7157895e+00	**0.0000000e+00**	5.5927579e+03	**0.0000000e+00**
AckleyTr	7.4604799e+00	**6.8938606e-10**	2.2437051e+00	**3.7213526e-03**
ElliptiqueTr	4.6097146e+06	**5.6578530e-14**	8.1945486e+06	**1.4601956e+01**
RastriginTr	4.9130236e+01	**2.2070429e+00**	2.5456579e+02	**7.0045334e-01**
RosenbrockTr	4.8627212e+02	**4.4709111e+01**	3.6494346e+04	**5.1927024e+02**
SphèreTr	1.4289255e-03	**1.4112576e-27**	3.3569831e+02	**6.1304737e-19**
AckleyRot	1.0530490e+01	**2.2013805e+00**	8.2996023e+00	**3.0771901e-03**
ElliptiqueRot	7.5605606e+05	**2.6553965e+05**	1.1848275e+07	**1.8099522e+06**
RastriginRot	**3.4530503e+01**	4.0322002e+01	**2.2569570e+02**	3.9389881e+02

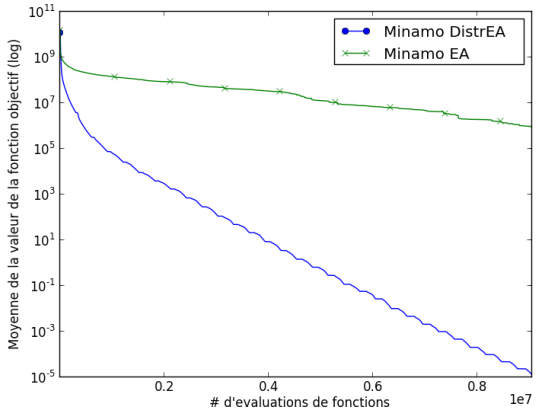
Table 5.5 – Solutions moyennes des algorithmes `Minamo EA` et `Minamo DistrEA`

liées les unes aux autres. Tout comme l'affirment également Yang, Tang et Yao dans [26], aucun algorithme distribué ne peut fournir une solution optimale pour de telles fonctions. En pratique, il est cependant très rare de faire face à des cas aussi extrêmes. Dans bon nombre d'applications, certaines variables peuvent être liées entre elles mais elles ne sont que rarement, voire jamais, toutes fortement liées. Enfin, la solution moyenne obtenue pour la fonction Rastrigin n'est, elle non plus, pas très précise. Cela est dû au fait que la fonction Rastrigin est fortement multimodale. Sur les 100 runs effectués, le minimum global est trouvé à plusieurs reprises mais il arrive également que l'algorithme converge vers un minimum local. Cela conduit à une solution moyenne peu précise.

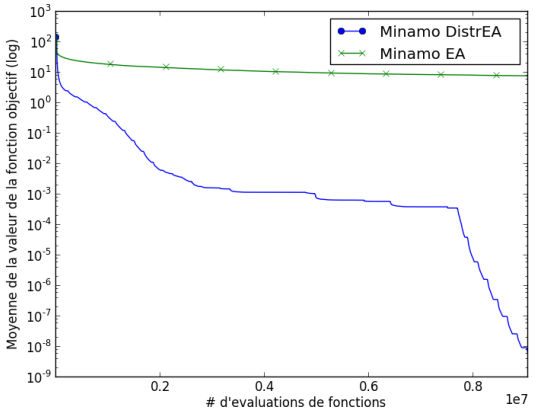
Pour plus de détails sur ces résultats, les tableaux statistiques, reprenant les performances de chacun des algorithmes, sont fournis en Annexe aux Tables A1 et A2.



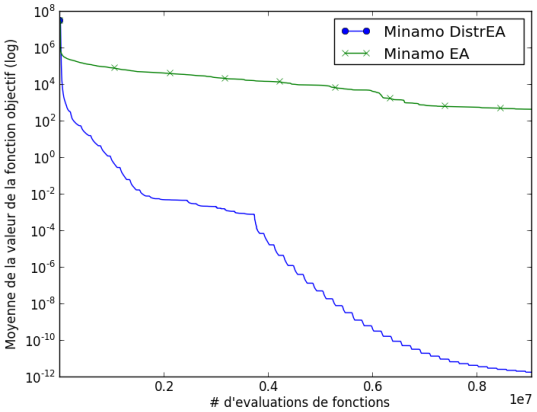
(a) Ackley - 500D



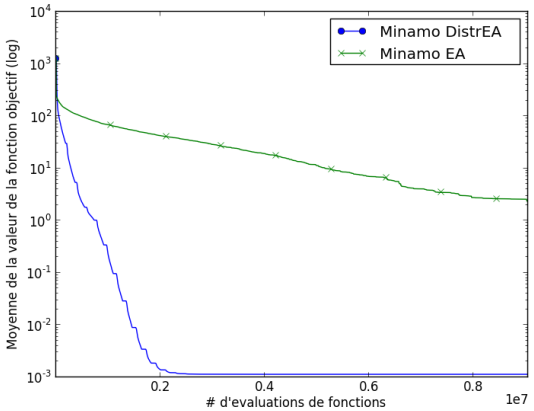
(b) Elliptique - 500D



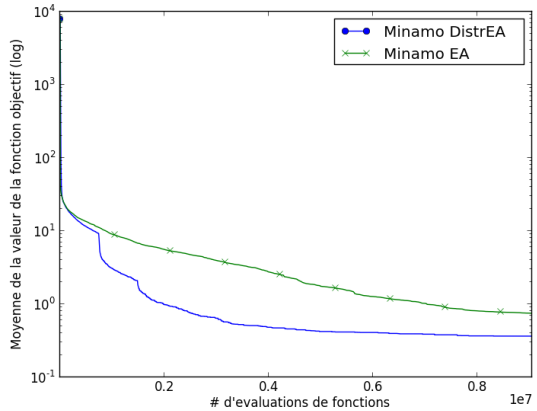
(c) Generalized Penalized 1 - 500D



(d) Generalized Penalized 2 - 500D

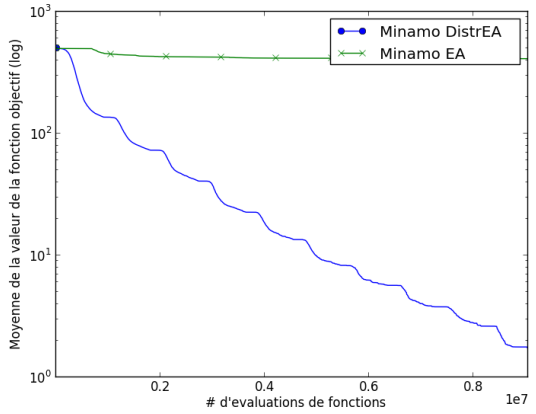


(e) Griewank - 500D

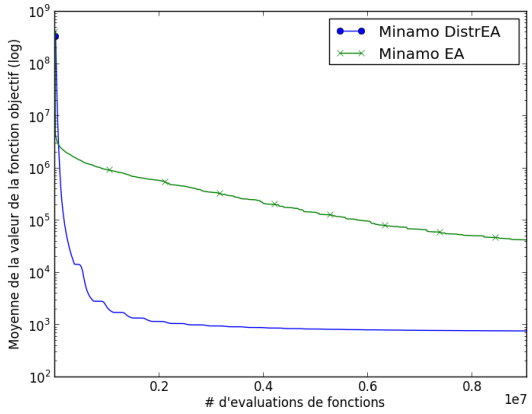


(f) Quartique - 500D

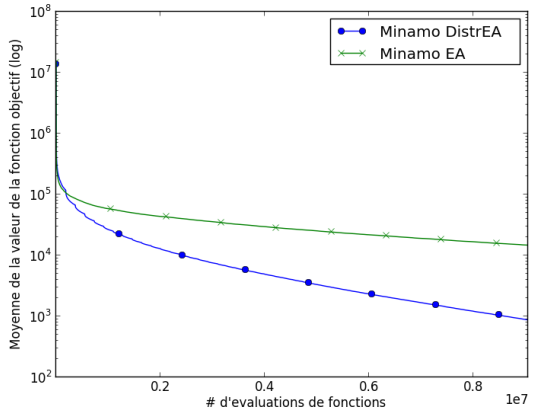
FIGURE 5.10 – Graphes de convergence des algorithmes `Minamo EA` et `Minamo DistrEA`



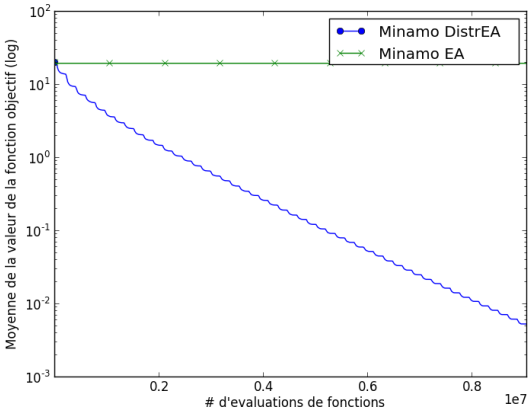
(a) Rastrigin - 500D



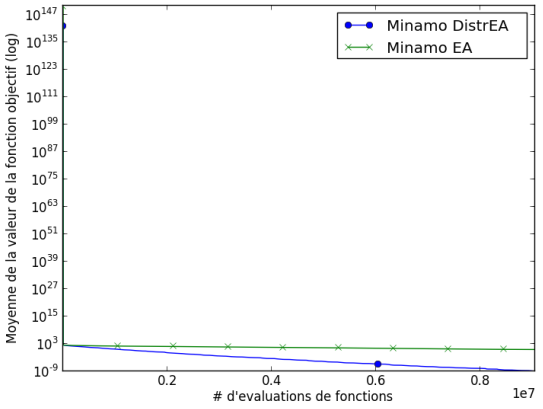
(b) Rosenbrock - 500D



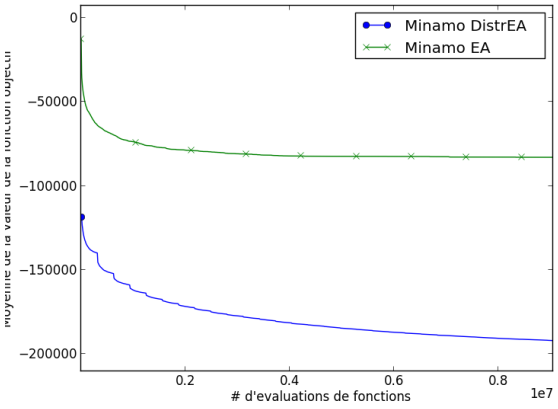
(c) Schwefel 12 - 500D



(d) Schwefel 2.21 - 500D

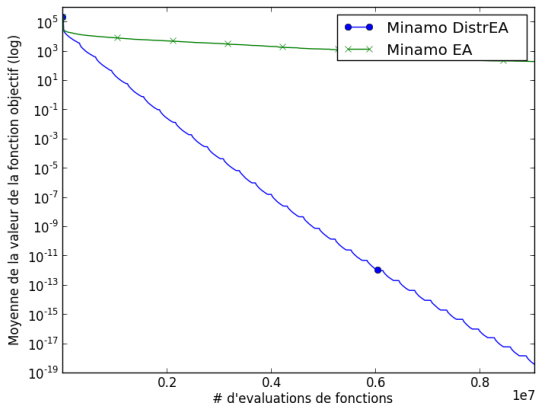


(e) Schwefel 2.22 - 500D

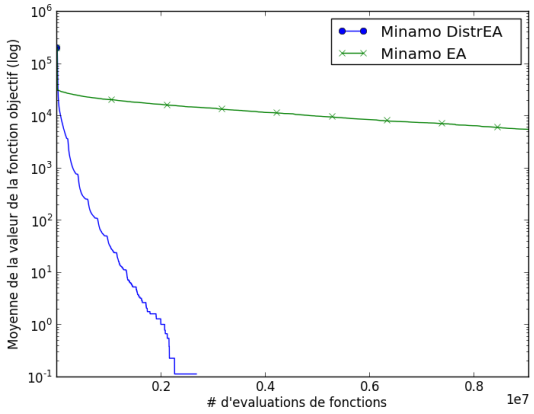


(f) Schwefel 2.26 - 500D

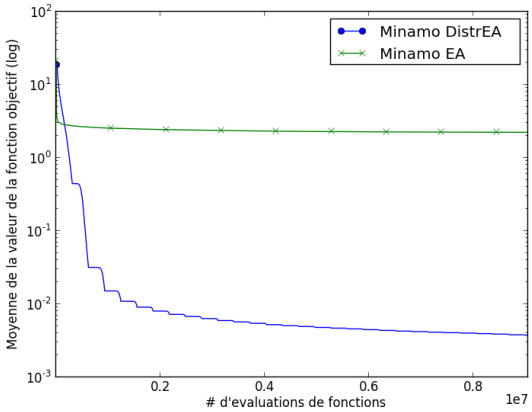
FIGURE 5.11 – Graphes de convergence des algorithmes `Minamo EA` et `Minamo DistrEA`



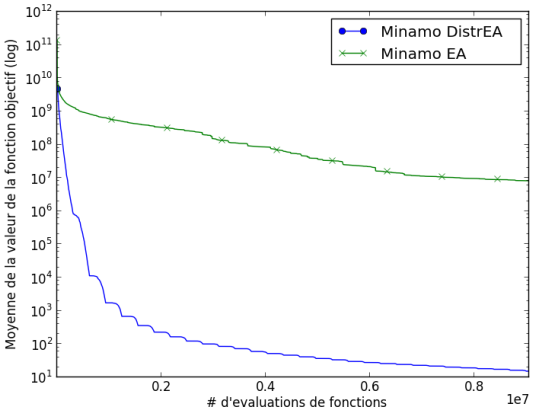
(a) Sphère - 500D



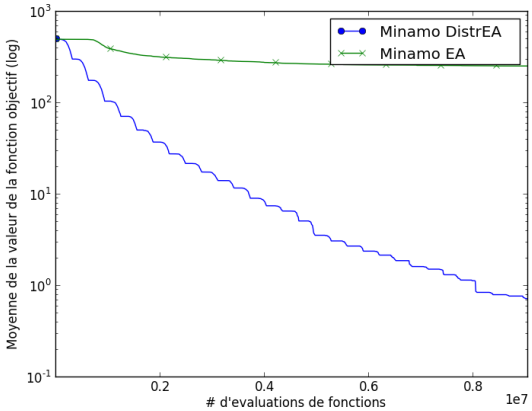
(b) Step - 500D



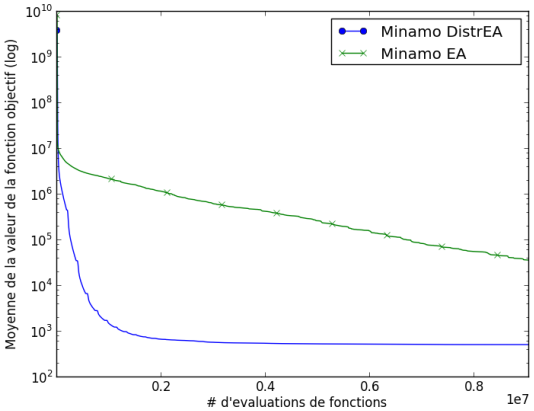
(c) AckleyTr - 500D



(d) ElliptiqueTr - 500D

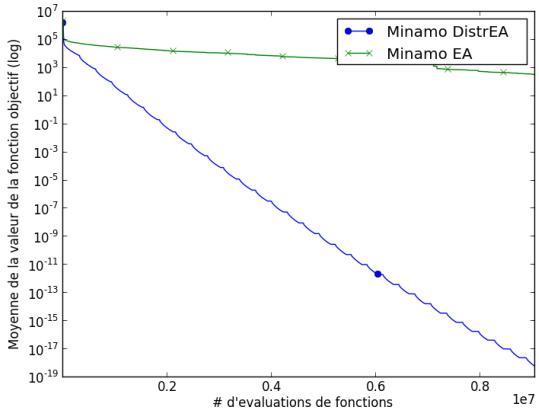


(e) RastriginTr - 500D

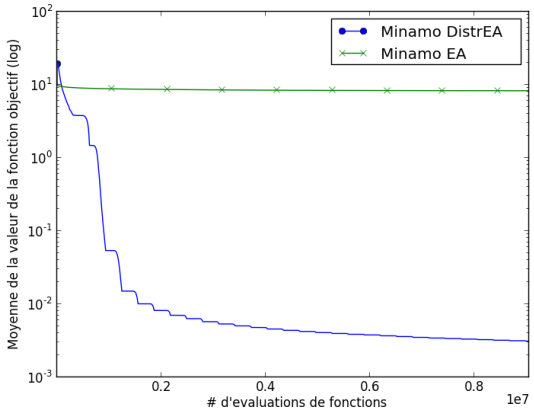


(f) RosenbrockTr - 500D

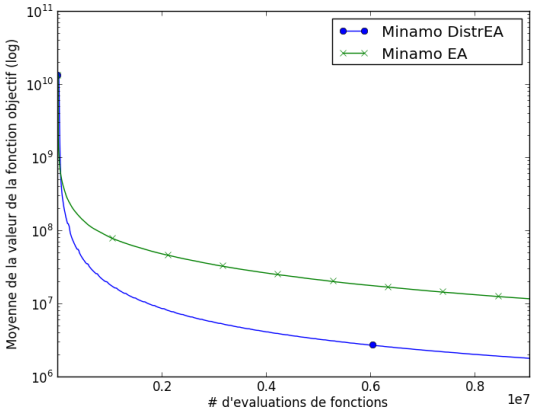
FIGURE 5.12 – Graphes de convergence des algorithmes `Minamo EA` et `Minamo DistrEA`



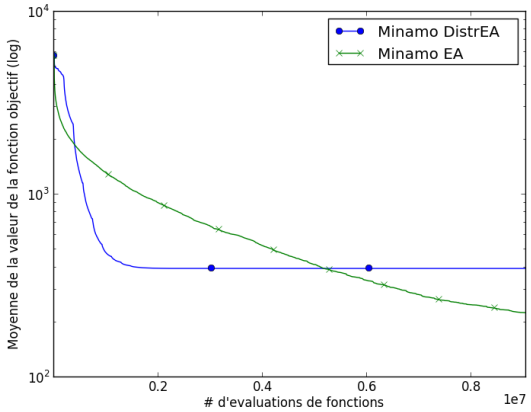
(a) SphèreTr - 500D



(b) AckleyRot - 500D



(c) ElliptiqueRot - 500D



(d) RastriginRot - 500D

Figure 5.13 – Graphes de convergence des algorithmes `Minamo EA` et `Minamo DistrEA`

Chapitre 6

Stratégie d'adaptation dynamique du nombre de sous-populations

Lors du chapitre précédent, nous avons présenté l'algorithme `Minamo DistrEA` et nous avons testé celui-ci sur une large gamme de fonctions test. Ces tests ont, notamment, mis en valeur l'importance du nombre de sous-populations choisi pour obtenir une solution relativement précise. En effet, pour bon nombre de fonctions, il est idéal de travailler avec un grand nombre de sous-populations en début d'optimisation, ceci dans le but de trouver une bonne région rapidement. Cependant, une fois cette bonne région trouvée, il est préférable de travailler avec un plus petit nombre de sous-populations de telle sorte que chacune d'entre elles contienne une grande proportion de l'information globale. Dans ce chapitre, nous développons un nouvel algorithme qui permet d'adapter le nombre de sous-populations en cours d'optimisation. Celui-ci est dénommé `Minamo SADistrEA` (pour *Minamo Self-Adaptive Distributed Evolutionary Algorithm*). Nous commençons par en décrire le principe et son implémentation dans Minamo[1]. Par la suite, nous le testons sur l'ensemble des fonctions test avec 50 et 500 variables et nous comparons les résultats obtenus avec ceux de l'algorithme `Minamo DistrEA`.

6.1 L'implémentation

L'idée principale de l'algorithme `Minamo SADistrEA` est d'adapter le nombre de sous-populations en cours d'optimisation. Comme nous l'avons présenté à la Section 3.1, ce concept a été développé par Omidvar, Li, Yang et Yao [21]. Pour rappel, leur méthode consiste à choisir aléatoirement un nouveau nombre de sous-populations lorsqu'aucune amélioration n'est obtenue d'un cycle à l'autre.

Dans notre cas, nous choisissons une approche quelque peu différente. Nous commençons l'optimisation avec un grand nombre de sous-populations. Ensuite, à chaque fois que l'amélioration d'un cycle à l'autre est jugée trop faible, nous diminuons le nombre

1. Pour des questions de confidentialité, nous ne pouvons présenter ici-même les codes que nous avons implémentés.

de sous-populations. Cette idée nous vient des conclusions tirées à la Section 5.2.2 selon lesquelles il est préférable d'avoir un grand nombre de sous-populations en début d'optimisation et un plus petit nombre par la suite.

Dans le but de décrire l'implémentation de ce nouvel algorithme, nous introduisons deux nouvelles notions.

Définition 6.1 (Suite des nombres de sous-populations)
*La **suite des nombres de sous-populations**, que nous notons $S = S_1, S_2, \ldots, S_s$ est une suite ordonnée d'entiers telle que $2 \leq S_1 < S_2 < \cdots < S_s \leq n$, où n désigne le nombre de variables de la fonction objectif.*

L'idée est de construire S_s sous-populations lors du premier cycle. Par la suite, lorsque nous souhaitons diminuer le nombre de sous-populations, nous en construisons S_{s-1}, puis S_{s-2} et ainsi de suite.

Définition 6.2 (Taux d'amélioration)
*Le **taux d'amélioration**, noté r, est défini par*

$$r = \frac{|f_c - f_p|}{|f_p|},$$

où f_c est le fitness du meilleur individu au cycle courant et f_p est le fitness du meilleur individu au cycle précédent.

Si le taux d'amélioration est proche de zéro, cela signifie que l'amélioration obtenue lors du cycle courant est très faible, voire nulle. Dans ce cas, nous diminuons le nombre de sous-populations pour le cycle suivant. Au contraire, si le taux d'amélioration est grand, cela signifie que le nombre de sous-populations choisi a permis d'obtenir un gain relativement grand et ce nombre est de nouveau utilisé pour le cycle suivant. Dans la suite, nous jugeons que le taux d'amélioration est proche de zéro lorsque celui-ci est plus petit qu'un certain seuil que nous notons ε. Ce dernier doit être strictement compris entre 0 et 1. Ainsi, si on choisit par exemple $\varepsilon = 0.5$, cela signifie que l'amélioration de f_c par rapport à f_p doit être d'au moins 50% pour que le taux d'amélioration soit considéré comme grand et que le nombre de sous-populations soit de nouveau utilisé au cycle suivant.

En plus d'adapter le nombre de sous-populations lors de chaque nouveau cycle, nous adaptons aussi le nombre de générations. En effet, lorsque le nombre de sous-populations est grand, cela signifie que chacune d'entre elles est caractérisée par un faible nombre de variables. Dans ce cas, peu de générations sont nécessaires pour que les individus de chacune des sous-populations convergent vers un même individu, considéré comme étant le meilleur. Cependant, lorsque le nombre de sous-populations est petit, chacune d'entre elles est caractérisée par un nombre important de variables et un plus grand nombre de générations est alors nécessaire pour que les individus de chacune des sous-populations convergent vers un même individu. Pour ce faire, nous fixons un nombre

d'évaluations de fonctions identiques pour chacun des cycles. Celui-ci, noté $FEpc$ (pour *Function evaluation per cycle*), est donné par

$$FEpc = (p \times m + 1) \times (maxGenerations + 1)$$

où p désigne le nombre d'individus de chacune des sous-populations et m est le nombre de sous-populations (pour plus de détails, voir Section 5.1). Ainsi, lorsque le nombre de sous-populations diminue, nous augmentons le nombre de générations de façon à réaliser le même nombre d'évaluations de fonctions lors de chaque cycle.

L'implémentation de l'algorithme `Minamo SADistrEA` est décrite à l'Algorithme 6.1. Celle-ci est en grande partie similaire à celle de l'algorithme `Minamo DistrEA` décrit à l'Algorithme 5.1. Afin de mettre en évidence les différences entre ces deux algorithmes, les nouvelles étapes propres à l'adaptation du nombre de sous-populations et de générations sont grisées.

6.2 La présentation des résultats

Nous testons à présent l'algorithme `Minamo SADistrEA` sur l'ensemble des fonctions décrites au Chapitre 4 pour 50 et 500 variables. Les performances de ce dernier sont comparées à celle de l'algorithme `Minamo DistrEA`. Les deux algorithmes sont comparés sur le même nombre d'évaluations de fonctions, 7×10^5 pour les fonctions à 50 variables et 9×10^6 pour les fonctions à 500 variables. De plus, pour chacun des algorithmes, la taille de la population est fixée à 35 individus en dimension 50 et à 100 individus en dimension 500. Les résultats présentés pour l'algorithme `Minamo DistrEA` sont identiques à ceux proposés au chapitre précédent et sont donc obtenus à partir des paramètres décrits à la Table 5.4, adaptés à chacune des fonctions. Quant aux paramètres de l'algorithme `Minamo SADistrEA`, nous fixons, suite à une recherche empirique, le nombre de cycles à 10 et le seuil du taux d'amélioration, ε, à 0.7. Notons que la valeur choisie pour ε peut paraitre grande car elle impose que le taux d'amélioration d'un cycle à l'autre doit être d'au moins 70% pour qu'il soit considéré comme grand. En pratique, il est nécessaire que cette valeur soit relativement grande. En effet, si elle est trop petite, le taux d'amélioration est presque toujours considéré comme grand. Cela implique qu'on ne diminue presque jamais le nombre de sous-populations et qu'on ne profite donc pas du caractère auto-adaptatif de l'algorithme `Minamo SADistrEA`. Quant aux suites de nombres de sous-populations, elles sont données respectivement par $S = \{2, 5, 10, 25\}$ et $S = \{2, 5, 10, 25, 50, 100, 250\}$ pour les fonctions à 50 et 500 variables. Contrairement au chapitre précédent, nous nous permettons de choisir un nombre relativement grand de sous-populations en début d'algorithme. Si celui-ci permet une grande amélioration, c'est tout à notre avantage, sinon on revient à un nombre de sous-populations plus conventionnel lors du deuxième cycle.

De nouveau, l'ensemble des tests sont réalisés sur 100 runs. À la Table 6.1, est proposée, pour chacune des fonctions, la solution moyenne obtenue par les algorithmes `Minamo DistrEA` et `Minamo SADistrEA`. Des graphes de convergence comparant les

Algorithme 6.1 : Minamo SADistrEA

Étape 1 :

1. Construire une population *pop* de p individus.
2. Évaluer *pop* et trouver le meilleur individu *best*.
3. Poser $r = 1$, $i = s$, $f_p = fitness(best)$ et $cycle = 0$.

Étape 2 :

1. Si $r < \varepsilon$ et $i > 1$,
   - Décrémenter i.
2. Poser $m = S_i$.
3. Séparer *pop* en m sous-populations $pop_1, \ldots, pop_m$.
4. Évaluer $pop_1, \ldots, pop_m$ à l'aide de *best*.
5. Extraire l'individu « représentant » *rep* des sous-populations $pop_1, \ldots, pop_m$.
6. Évaluer *rep*.
7. Poser $FE = p \times m + 1$.

Étape 3 :

1. Appliquer les opérateurs génétiques à $pop_1, \ldots, pop_m$.
2. Évaluer $pop_1, \ldots, pop_m$ à l'aide de *rep*.
3. Extraire le nouvel individu « représentant » *rep* des sous-populations $pop_1, \ldots, pop_m$.
4. Évaluer *rep*.
5. Mettre-à-jour $FE : FE = FE + p \times m + 1$.
6. Si $fitness(rep) < fitness(best)$,
   $best = rep$.
7. Si $FE \leq FEpc - (p \times m + 1)$,
   - Aller à l'étape 3.1.

Étape 4 :

1. Si $cycle < maxCycles$,
   - Incrémenter *cycle*.
   - Poser $f_c = fitness(best)$.
   - Mettre-à-jour $r : r = \frac{|f_c - f_p|}{|f_p|}$.
   - Poser $f_p = f_c$.
   - Construire une population *pop* de p individus contenant l'individu *best*.
   - Aller à l'étape 2.

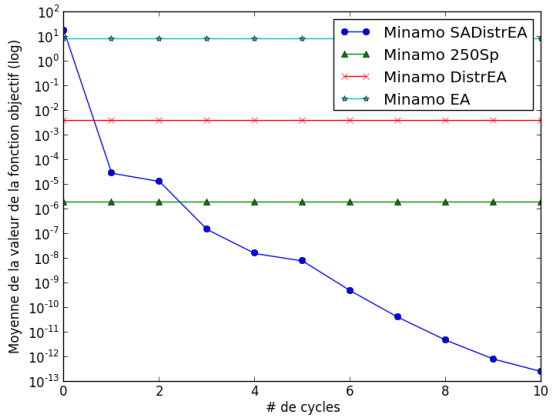
   Sinon,
   - Fin de l'optimisation.
   - Solution = $fitness(best)$.

algorithmes `Minamo EA`, `Minamo DistrEA` et `Minamo SADistrEA` pour les fonctions à 500 variables sont également fournis aux FIGURES 6.1 à 6.4. Notons que ces graphes sont quelque peu différents de ceux présentés précédemment. Ils représentent l'évolution de la moyenne de la valeur de la fonction objectif en fonction du nombre de cycles (et non en fonction du nombre d'évaluations de fonctions) pour l'algorithme `Minamo SADistrEA`. Les solutions moyennes obtenues par les algorithmes `Minamo EA` et `Minamo DistrEA` sont quant à elles représentées au moyen de lignes horizontales.

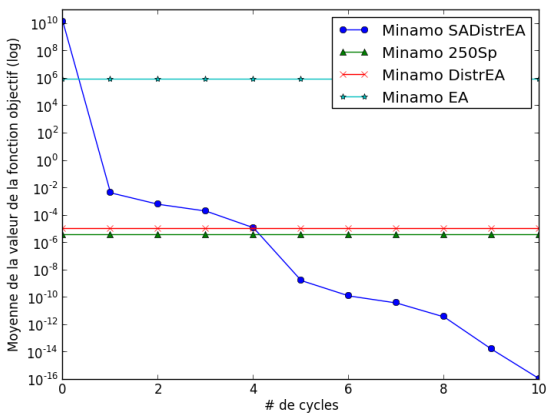
En analysant les résultats, nous remarquons que pour neuf des fonctions testées, à savoir les fonctions Ackley, GeneralizedPenalized 1, Griewank, Rastrigin, Rosenbrock, Schwefel 2.26, AckleyTr, ElliptiqueTr et RastriginTr, la meilleure performance obtenue par l'algorithme `Minamo SADistrEA` semble être due, en partie, à l'utilisation de 250 sous-populations en début d'algorithme. En effet, pour ces fonctions, nous pouvons voir sur les différents graphes de convergence que l'algorithme `Minamo SADistrEA` est déjà meilleur que l'algorithme `Minamo DistrEA` après seulement un cycle. Afin de prouver que le grand nombre de sous-populations n'est pas l'unique raison de ces meilleures performances, nous représentons également, sur chacun des graphes de convergence, les solutions moyennes obtenues avec l'algorithme `Minamo DistrEA` dont le nombre de sous-populations est fixé à 250 et le nombre de cycle à 10. Ce dernier est dénommé `Minamo 250Sp`.

	50D-DistrEA	50D-SADistrEA	500D-DistrEA	500D-SADistrEA
Ackley	3.8659919e-10	**7.3625316e-15**	3.8251430e-03	**2.6005397e-13**
Elliptique	1.0685601e-22	**1.2708699e-38**	1.1389644e-05	**1.1647897e-16**
GenPen1	2.3746101e-26	**9.4232686e-33**	3.0736311e-09	**3.8904831e-25**
GenPen2	1.6373381e-23	**1.3497838e-32**	1.7608379e-12	**1.0732272e-22**
Griewank	1.2874647e-02	**3.0590474e-03**	1.1422252e-03	**2.0918939e-16**
Quartique	**3.7719892e-02**	4.5464758e-02	2.9938697e-01	**2.9350369e-01**
Rastrigin	1.4348359e+00	**4.6082314e-01**	1.6927300e+00	**1.3163739e-14**
Rosenbrock	4.4657126e+01	**3.7861053e+01**	7.1871933e+02	**1.0324862e+01**
Schwefel 12	**1.1284238e-03**	2.3285494e+00	**8.7209365e+02**	1.5553263e+04
Schwefel 2.21	8.5391820e-08	**3.8736625e-10**	**4.7193498e-03**	3.0406840e-01
Schwefel 2.22	2.4876855e-21	**2.7886295e-25**	1.1583291e-09	**1.1356821e-12**
Schwefel 2.26	-1.9404062e+04	**-2.0262042e+04**	-1.9225367e+05	**-2.0651095e+05**
Sphère	6.9512707e-28	**2.6889505e-41**	3.9208333e-19	**3.2558977e-21**
Step	**0.0000000e+00**	**0.0000000e+00**	**0.0000000e+00**	**0.0000000e+00**
AckleyTr	6.8938606e-10	**7.0633557e-15**	3.7213526e-03	**3.1177400e-13**
ElliptiqueTr	5.6578530e-14	**1.2977904e-36**	1.4601956e+01	**3.4438116e-15**
RastriginTr	2.2070429e+00	**4.8177011e-01**	7.0045334e-01	**0.0000000e+00**
RosenbrockTr	**4.4709111e+01**	6.4247288e+01	**5.1927024e+02**	6.2843666e+02
SphèreTr	1.4112576e-27	**9.8492062e-41**	6.1304737e-19	**1.2792947e-20**
AckleyRot	2.2013805e+00	**7.0259588e-15**	3.0771901e-03	**6.0964099e-13**
ElliptiqueRot	**2.6553965e+05**	3.8072797e+07	**1.8099522e+06**	5.2465463e+06
RastriginRot	**4.0322002e+01**	9.9859252e+01	**3.9389881e+02**	6.8350484e+02

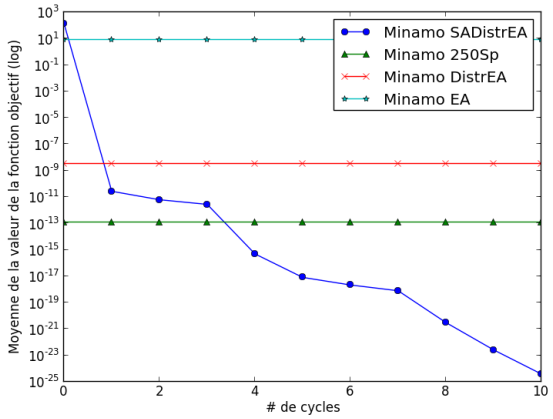
TABLE 6.1 – Solutions moyennes des algorithmes `Minamo DistrEA` et `Minamo SADistrEA`



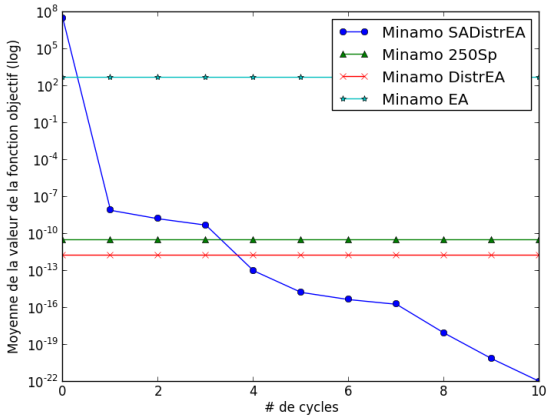
(a) Ackley - 500D



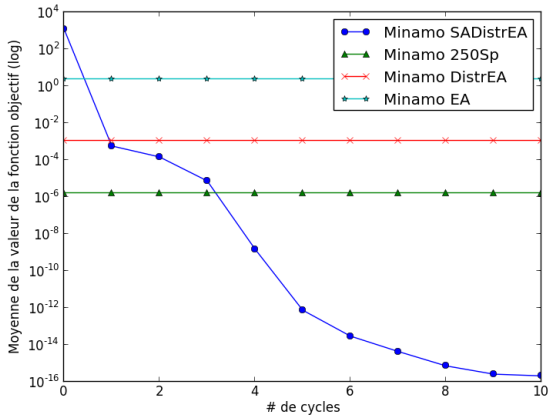
(b) Elliptique - 500D



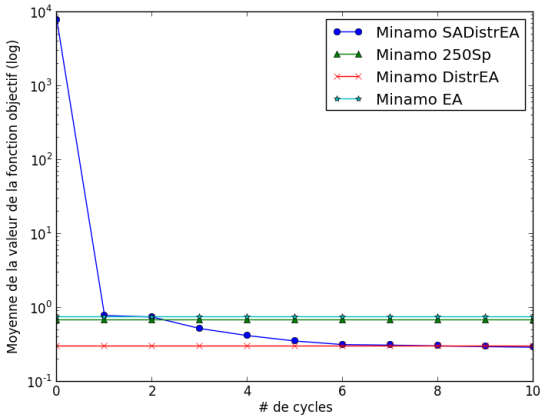
(c) Generalized Penalized 1 - 500D



(d) Generalized Penalized 2 - 500D

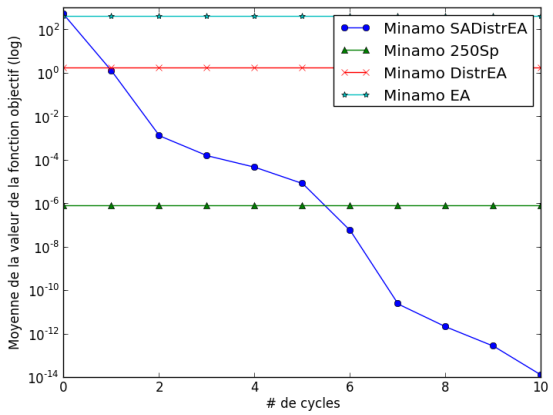


(e) Griewank - 500D

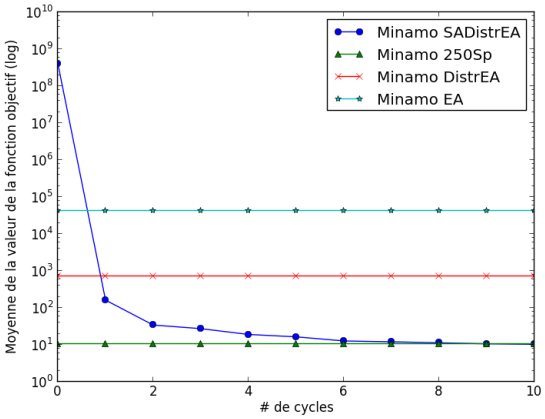


(f) Quartique - 500D

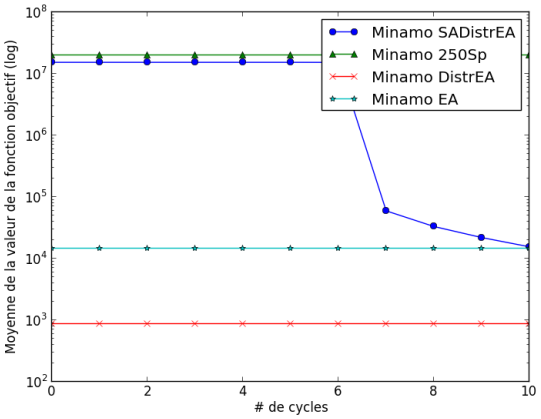
FIGURE 6.1 – Graphes de convergence des algorithmes `Minamo EA`, `Minamo DistrEA`, `Minamo 250Sp` et `Minamo SADistrEA`



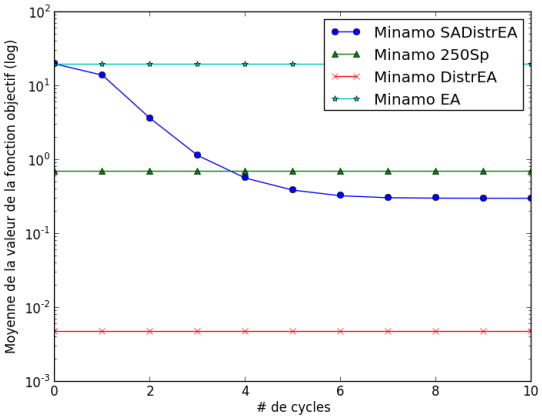
(a) Rastrigin - 500D



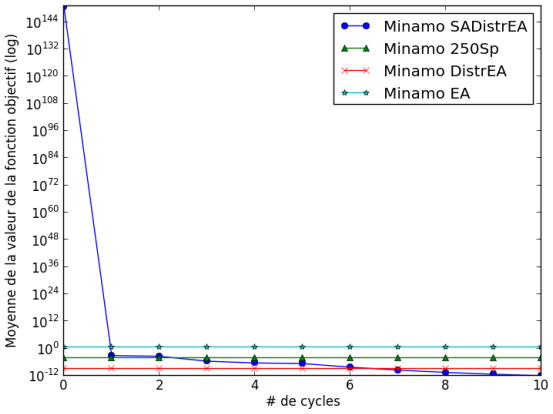
(b) Rosenbrock - 500D



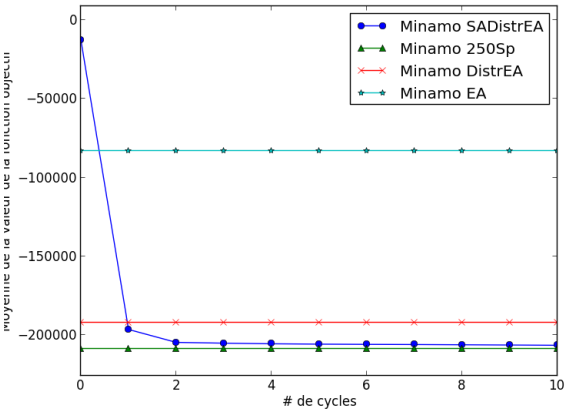
(c) Schwefel 12 - 500D



(d) Schwefel 2.21 - 500D

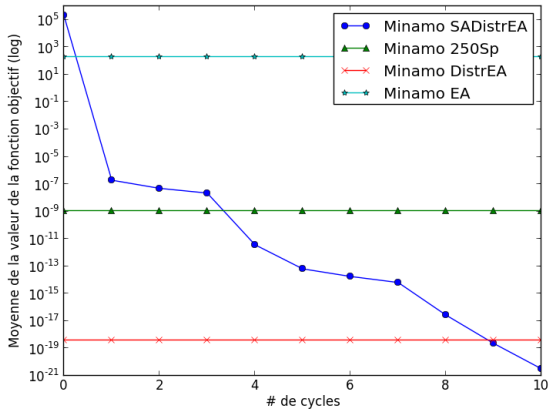


(e) Schwefel 2.22 - 500D

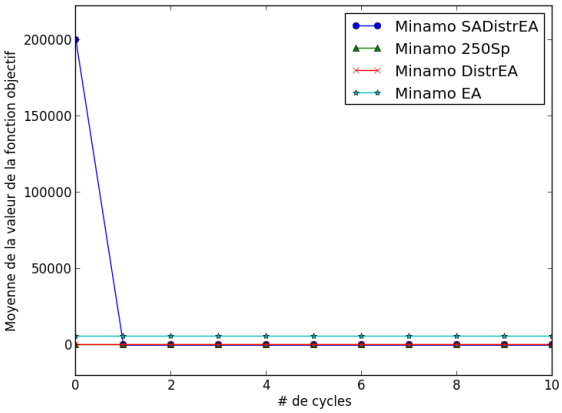


(f) Schwefel 2.26 - 500D

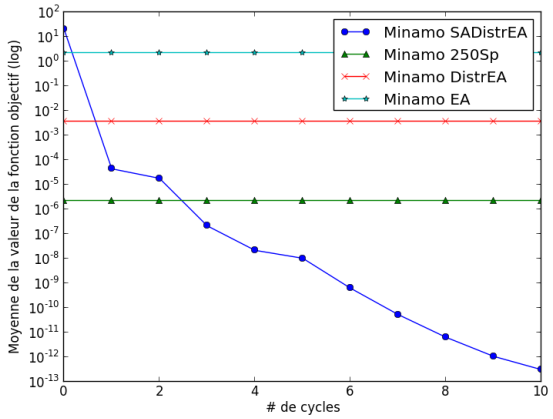
FIGURE 6.2 – Graphes de convergence des algorithmes `Minamo EA`, `Minamo DistrEA`, `Minamo 250Sp` et `Minamo SADistrEA`



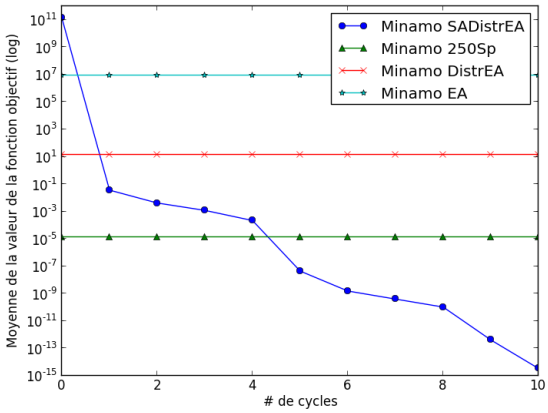
(a) Sphère - 500D



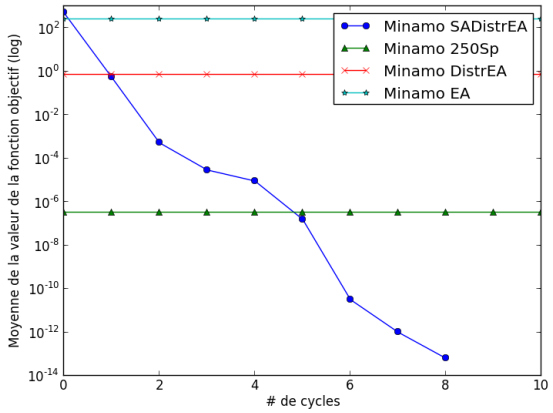
(b) Step - 500D



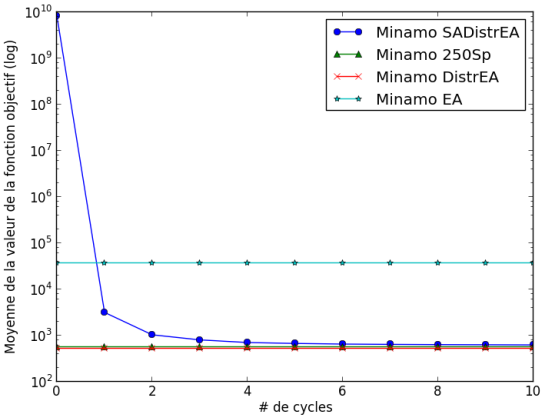
(c) AckleyTr - 500D



(d) ElliptiqueTr - 500D

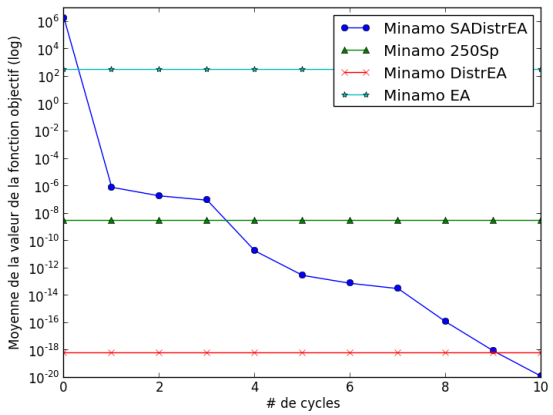


(e) RastriginTr - 500D

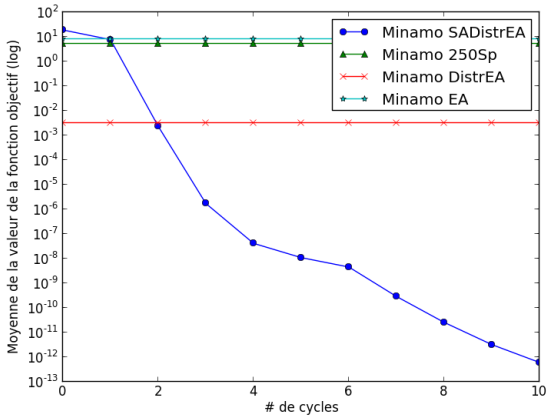


(f) RosenbrockTr - 500D

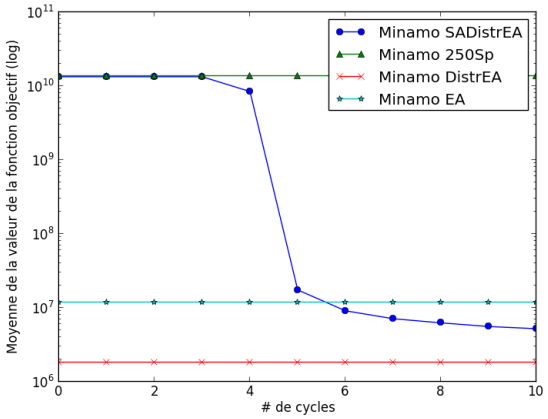
FIGURE 6.3 – Graphes de convergence des algorithmes `Minamo EA`, `Minamo DistrEA`, `Minamo 250Sp` et `Minamo SADistrEA`



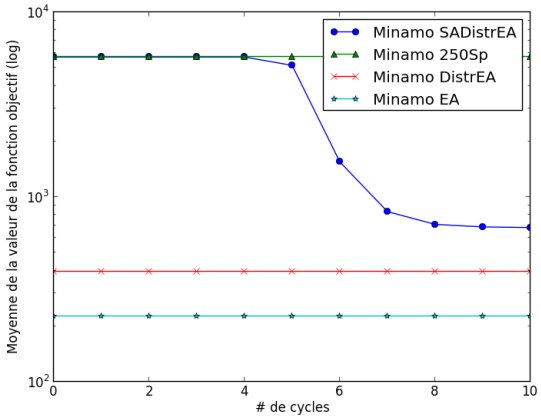
(a) SphèreTr - 500D



(b) AckleyRot - 500D



(c) ElliptiqueRot - 500D



(d) RastriginRot - 500D

FIGURE 6.4 – Graphes de convergence des algorithmes `Minamo EA`, `Minamo DistrEA`, `Minamo 250Sp` et `Minamo SADistrEA`

Le nouvel algorithme `Minamo SADistrEA` permet d'obtenir de meilleurs résultats que `Minamo 250Sp` sur toutes les fonctions, à l'exception des fonctions Schwefel 2.26 et RosenbrockTr pour lesquelles les résultats obtenus par ces deux algorithmes sont relativement proches. Par ailleurs, l'algorithme `Minamo SADistrEA` est meilleur que l'algorithme `Minamo DistrEA` sur la plupart des fonctions, les fonctions faisant exception étant les fonctions Schwefel 12, Schwefel 2.21, RosenbrockTr, ElliptiqueRot et RastriginRot. Celles-ci sont assez particulières car, afin de les optimiser de manière efficace, il est nécessaire de travailler avec un nombre de sous-populations relativement faible dès le début de l'optimisation. En effet, les fonctions Schwefel 12, Schwefel 2.21 et RosenbrockTr, de par leur expression analytique, et les fonctions ElliptiqueRot et RastriginRot, de par la rotation appliquée à l'espace des variables, sont des fonctions à caractère non séparable pour lesquelles toutes les variables sont fortement liées les unes

aux autres. Dès lors, pour de telles fonctions, il est nécessaire de travailler avec peu de sous-populations de façon à ce que de nombreuses variables soient groupées au sein d'une même sous-population. Lorsqu'on applique l'algorithme `Minamo SADistrEA` à de telles fonctions, on dépense de nombreuses ressources sur un nombre de sous-populations élevé en début d'optimisation sans obtenir aucune amélioration. En effet, au début, les grands nombres de sous-populations ne permettent pas une décroissance de la valeur de la fonction objectif. Comme nous pouvons le voir sur les graphes de convergence, cette valeur stagne jusqu'au moment où elle chute brusquement. Il s'agit du moment où le nombre de sous-populations est devenu suffisamment petit. Par exemple, pour la fonction Schwefel 12 (FIGURE 6.2c), cette brusque chute a lieu lors du cycle 7, cycle lors duquel le nombre de sous-populations est auto-adapté à 2 tandis que pour la fonction ElliptiqueRot (FIGURE 6.4c), elle a lieu lors du cycle 5, cycle lors duquel le nombre de sous-populations est auto-adapté à 10.

Pour ce qui est des fonctions pour lesquelles l'algorithme `Minamo SADistrEA` offre de meilleurs résultats, il est possible de lire, sur leur graphe de convergence, les nettes améliorations dues à l'auto-adaptation du nombre de sous-populations en cours d'optimisation. Pour la fonction Ackley par exemple (FIGURE 6.5), le premier cycle, réalisé avec 250 sous-populations, permet une grande amélioration de la valeur de la fonction objectif. Le deuxième cycle est alors exécuté en conservant 250 sous-populations. Cette fois, le taux d'amélioration est relativement faible et le nombre de sous-populations passe à 100 pour le troisième cycle. Cela conduit à une amélioration conséquente qui permet d'obtenir, dès le troisième cycle, une meilleure solution que si nous avions conservé 250 sous-populations lors de toute l'optimisation. Notons qu'une accélération de la convergence est également observée lors du sixième cycle. Il s'agit du moment où le nombre de sous-populations est diminué de 100 à 50. D'une manière générale, nous pouvons dire que lorsque la pente de la courbe de convergence devient significativement plus forte qu'auparavant, cela correspond à une diminution du nombre de sous-populations qui a des effets très bénéfiques dans le processus de convergence vers la solution.

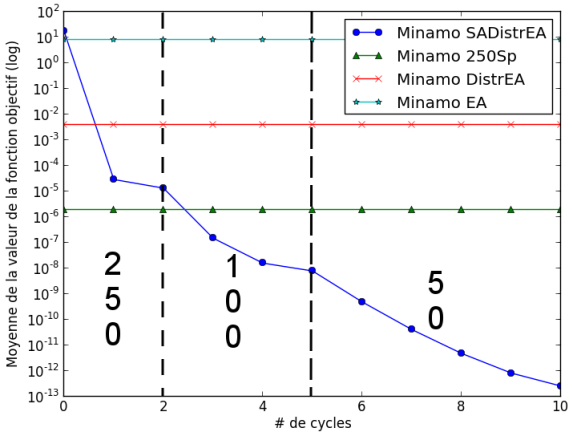


FIGURE 6.5 – Adaptation du nombre de sous-populations pour Ackley - 500D

Pour plus de détails sur ces résultats, un tableau statistique, reprenant les performances de l'algorithme `Minamo SADistrEA`, est fourni en Annexe à la Table A3. Les nombres dans les colonnes « Meilleur » et « Pire » correspondent aux nombres de sous-populations, en fin d'optimisation, obtenus lors du meilleur et du pire run.

En conclusion, nous venons de montrer l'efficacité de l'auto-adaptation du nombre de sous-populations en cours d'optimisation. Cette nouvelle fonctionnalité offre un avantage double. D'une part, l'algorithme `Minamo SADistrEA` offre de meilleurs résultats que l'algorithme `Minamo DistrEA`. D'autre part, l'impact des choix des paramètres sur les performances de ce nouvel algorithme est moindre que pour l'algorithme `Minamo DistrEA`. En effet, pour obtenir de bonnes performances avec ce dernier, il est nécessaire de choisir des paramètres en adéquation avec le type de fonction tandis que pour l'algorithme `Minamo SADistrEA`, nous avons fixé les paramètres de manière identique pour toutes les fonctions testées et cela conduit à des résultats très satisfaisants.

Chapitre 7

Problèmes avec contraintes

Jusqu'à présent, nous nous sommes uniquement intéressés à des problèmes d'optimisation sans contraintes. Toutefois, dans le secteur du génie industriel, bon nombre de problèmes font intervenir une ou plusieurs contraintes. Dès lors, dans l'optique de pouvoir appliquer les algorithmes développés précédemment à des problèmes concrets, ceux-ci doivent pouvoir gérer les problèmes avec contraintes. Dans ce chapitre, nous abordons la gestion des contraintes par les algorithmes évolutionnaires et plus précisément par l'algorithme de MINAMO. Ensuite, nous présentons quelques problèmes sur lesquels nous allons tester l'algorithme `Minamo SADistrEA` et enfin, nous présentons les résultats obtenus.

7.1 La gestion des contraintes

Il existe de nombreuses méthodes qui permettent de gérer des problèmes avec contraintes à l'aide d'un algorithme évolutionnaire. Dans cette section, nous décrivons, sur base des références [7] et [16], quelques-unes des méthodes les plus couramment utilisées. Par la suite, nous abordons la gestion des contraintes par le logiciel MINAMO.

La méthode dite de la « peine de mort » est sans doute la plus simple qui permette de gérer les contraintes. Elle consiste à rejeter systématiquement les individus non admissibles. Elle fonctionne plutôt bien lorsque l'espace admissible est convexe et constitue une grande proportion de l'espace de recherche mais est sujette à de sérieuses limitations lorsque l'espace admissible est plus complexe.

Une autre approche, bien plus répandue et plus efficace, est la pénalisation des individus non admissibles. Le principe est de transformer un problème avec contraintes en un problème sans contraintes. On ajoute, à la fonction objectif, une fonction de pénalité qui est nulle aux points admissibles et strictement positive aux points non admissibles. Ces fonctions de pénalité sont multipliées par des constantes positives qui permettent de donner un poids aux différentes contraintes par rapport à la fonction objectif. Pour les pénalités dites « statiques », ces poids sont fixés arbitrairement, pour les pénalités dites « dynamiques », ils prennent en compte le numéro de la génération

en cours tandis que pour les pénalités dites « adaptatives », ils utilisent l'information du processus de recherche réalisé au préalable. Toutefois, cette méthode de pénalisation est sujette à un inconvénient de taille. Elle emploie de nombreux paramètres et il est très difficile de choisir les valeurs de ceux-ci. Il n'existe pas de règle générale afin de les déterminer.

Il est également possible de maintenir l'admissibilité des individus d'une population au moyen d'opérateurs génétiques spécifiques. Ceux-ci sont construits de telle sorte que la mutation d'un individu admissible donne un nouvel individu admissible et qu'un individu issu de la recombinaison de deux individus admissibles soit lui aussi admissible.

Une autre possibilité est de transformer le problème contraint en un problème multi-objectif non contraint. Pour ce faire, on construit un vecteur $\vec{v} = (f, f_1, \ldots, f_m)$ où f désigne la fonction objectif et f_i est une mesure de la violation de la contrainte i ($i = 1, \ldots, m$). Un algorithme évolutionnaire multi-objectif est alors appliqué pour minimiser les différentes composantes de ce vecteur.

La dernière approche que nous présentons est le tournoi de Deb [9]. Celui-ci propose d'appliquer la règle suivante pour comparer deux individus : lorsque deux individus s'affrontent, l'individu admissible est préféré à l'individu non admissible ; s'ils sont tous les deux admissibles, celui qui possède le meilleur fitness est préféré ; s'ils sont tous les deux non admissibles, celui qui viole le moins les contraintes est préféré. Cette approche possède l'avantage de ne pas nécessiter l'usage de paramètres et la règle de sélection qui privilégie toujours les individus admissibles permet de converger vers une solution admissible. Cependant, elle a aussi tendance à converger rapidement vers un optimum local lorsque la population initiale ne recouvre pas correctement tout l'espace.

Dans Minamo, différentes méthodes sont implémentées afin de gérer les contraintes. On y retrouve notamment différents types de pénalités et le tournoi de Deb. Celles-ci sont intégrées à Minamo au moyen de l'évaluation, pour chacun des individus, d'un objectif global qui prend en compte à la fois l'objectif et les contraintes. Dans ce mémoire, le mécanisme que nous utilisons afin de traiter les problèmes contraints est le tournoi de Deb, stratégie la plus couramment utilisée à Cenaero.

7.2 Les problèmes test

Dans cette section, nous présentons les problèmes d'optimisation avec contraintes sur lesquels nous allons tester l'algorithme `Minamo SADistrEA`. Les fonctions SphèreMOD et RosenbrockMOD sont les fonctions Sphère et Rosenbrock auxquels nous avons ajouté une contrainte d'inégalité. Quant aux fonctions G2, G3MOD et G19, il s'agit de problèmes contraints bien connus de la littérature, couramment utilisés au sein de l'équipe Minamo pour tester de nouvelles fonctionnalités du logiciel.

• **SphèreMOD**
L'expression du problème SphèreMOD est donnée par :

minimiser

$$f(x) = \sum_{i=1}^{n} x_i^2,$$

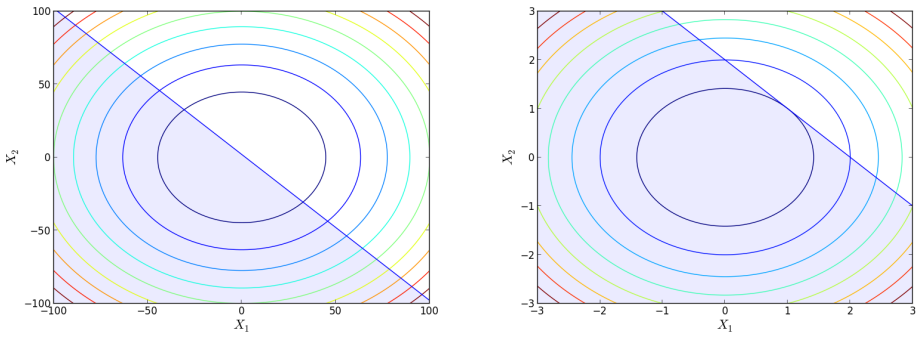
sous contrainte

$$g(x) = \sum_{i=1}^{n} x_i \geq n,$$

où n est le nombre de variables et $x \in [-100, 100]^n$. La solution est située au point

$$x_i^\star = 1, \qquad i = 1, \ldots, n$$

et $f(x^\star) = n$. Une représentation des courbes de niveau de la fonction f ainsi que de la zone admissible est proposée à la FIGURE 7.1a. Un zoom est également fourni à la FIGURE 7.1b.



(a) $x \in [-100, 100]^2$ (b) $x \in [-3, 3]^2$

FIGURE 7.1 – Courbes de niveau et zone admissible (en blanc) du problème SphereMOD

• **RosenbrockMOD**
L'expression du problème RosenbrockMOD est donnée par :

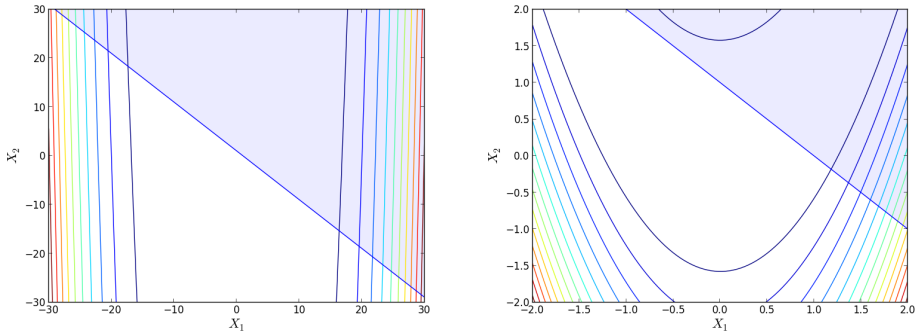
minimiser

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2],$$

sous contrainte

$$g(x) = \sum_{i=1}^{n} x_i \leq \tfrac{n}{2},$$

où n est le nombre de variables et $x \in [-30, 30]^n$. La meilleure solution connue [1] est $f(x^\star) = 12.229690$ pour $n = 50$ et $f(x^\star) = 124.74105$ pour $n = 500$. Une représentation des courbes de niveau de la fonction f ainsi que de la zone admissible est proposée à la FIGURE 7.2a. Un zoom est également fourni à la FIGURE 7.2b.



(a) $x \in [-30, 30]^2$ (b) $x \in [-2, 2]^2$

FIGURE 7.2 – Courbes de niveau et zone admissible (en blanc) du problème RosenbrockMOD

• **G2**

L'expression du problème G2 est donnée par :

minimiser

$$f(x) = -\left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|,$$

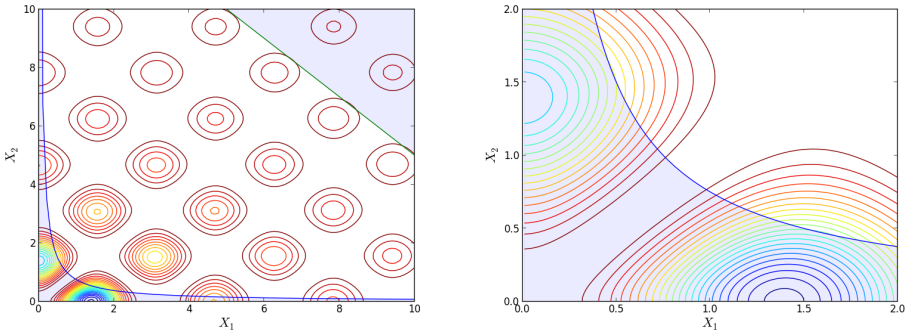
sous contraintes

$$g_1(x) = 0.75 - \prod_{i=1}^{n} x_i \leq 0,$$

$$g_2(x) = \sum_{i=1}^{n} x_i - 7.5n \leq 0,$$

où n est le nombre de variables et $x \in [0, 10]^n$. La meilleure solution connue pour $n = 20$ est $f(x^\star) = -0.803619$. Une représentation des courbes de niveau de la fonction f ainsi que de la zone admissible est proposée à la FIGURE 7.3a. Un zoom est également fourni à la FIGURE 7.3b.

1. Obtenue grâce à un algorithme de la classe des méthodes de points intérieurs de l'OPTIMIZATION TOOLBOX de MATLAB.



(a) $x \in [0, 10]^2$ (b) $x \in [0, 2]^2$

FIGURE 7.3 – Courbes de niveau et zone admissible (en blanc) du problème G2

• **G3MOD**
L'expression du problème G3MOD est donnée par :

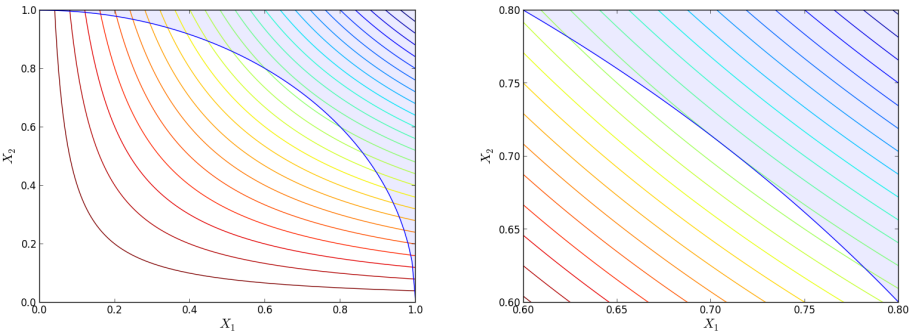
minimiser

$$f(x) = -\left(\sqrt{n}\right)^n \prod_{i=1}^{n} x_i,$$

sous contrainte

$$g_1(x) = \sum_{i=1}^{n} x_i^2 - 1 \leq 0,$$

où n est le nombre de variables et $x \in [0, 1]^n$. La meilleure solution connue pour $n = 20$ est $f(x^\star) = -0.9937$. Une représentation des courbes de niveau de la fonction f ainsi que de la zone admissible est proposée à la FIGURE 7.4a. Un zoom est également fourni à la FIGURE 7.4b.



(a) $x \in [0, 1]^2$ (b) $x \in [0.6, 0.8]^2$

FIGURE 7.4 – Courbes de niveau et zone admissible (en blanc) du problème G3MOD

• **G19**
L'expression du problème G19 est donnée par :

minimiser

$$f(x) = \sum_{i=1}^{5}\sum_{j=1}^{5} c_{ij}x_{10+i}x_{10+j} + 2\sum_{j=1}^{5} d_j x_{10+j}^3 - \sum_{i=1}^{10} b_i x_i,$$

sous contraintes

$$g_j(x) = -2\sum_{i=1}^{5} c_{ij}x_{10+i} - 3d_j x_{10+j}^2 - e_j + \sum_{i=1}^{10} a_{ij}x_i \leq 0 \ \forall j = 1..5,$$

où $x \in [0, 10]^{15}$,

$$b = [-40, -2, -0.25, -4, -4, -1, -40, -60, 5, 1]$$

et où les autres paramètres sont donnés à la TABLE 7.1.

j	1	2	3	4	5
e_j	-15	-27	-36	-18	-12
c_{1j}	30	-20	-10	32	-10
c_{2j}	-20	39	-6	-31	32
c_{3j}	-10	-6	10	-6	-10
c_{4j}	32	-31	-6	39	-20
c_{5j}	-10	32	-10	-20	30
d_j	4	8	10	6	2
a_{1j}	-16	2	0	1	0
a_{2j}	0	-2	0	0.4	2
a_{3j}	-3.5	0	2	0	0
a_{4j}	0	-2	0	-4	-1
a_{5j}	0	-9	-2	1	-2.8
a_{6j}	2	0	-4	0	0
a_{7j}	-1	-1	-1	-1	-1
a_{8j}	-1	-2	-3	-2	-1
a_{9j}	1	2	3	4	5
a_{10j}	1	1	1	1	1

TABLE 7.1 – Valeurs des paramètres pour le problème G19.

La meilleure solution connue est $f(x^\star) = 32.65559$.

7.3 La présentation des résultats

Nous comparons à présent les performances de l'algorithme `Minamo SADistrEA` à celles de l'algorithme `Minamo EA`. Les paramètres de ces algorithmes pour les différents problèmes testés sont repris à la TABLE 7.2. Les colonnes *Eval*, *Gen*, *Cyc*, *Pop* et ε y désignent respectivement le nombre d'évaluations de fonctions, de générations, de cycles, la taille de la population et le seuil du taux d'amélioration.

		`Minamo EA`		`Minamo SADistrEA`			
Problèmes	Eval	Gen	Pop	Suite des # de sous-populations	ε	Cyc	Pop
G2 - 20D	7×10^5	20 000	35	$S = \{2, 5, 10\}$	0.7	10	35
G3MOD - 20D	7×10^5	20 000	35	$S = \{2, 5, 10\}$	0.7	10	35
G19 - 15D	7×10^5	20 000	35	$S = \{2, 3, 7\}$	0.7	10	35
RosenbrockMOD - 50D	7×10^5	20 000	35	$S = \{2, 5, 10, 25\}$	0.7	10	35
RosenbrockMOD - 500D	9×10^6	90 000	100	$S = \{2, 5, 10, 25, 50, 100, 250\}$	0.7	10	100
SphèreMOD - 50D	7×10^5	20 000	35	$S = \{2, 5, 10, 25\}$	0.7	10	35
SphèreMOD - 500D	9×10^6	90 000	100	$S = \{2, 5, 10, 25, 50, 100, 250\}$	0.7	10	100

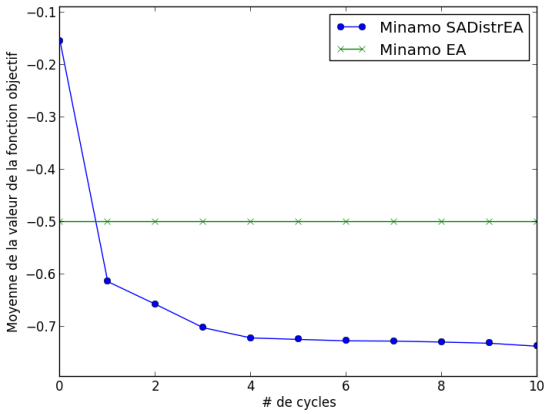
TABLE 7.2 – Paramètres des algorithmes `Minamo EA` et `Minamo SADistrEA`

De nouveau, l'ensemble des tests sont réalisés sur 100 runs. À la TABLE 7.3 est proposée, pour chacun des problèmes testés, la solution moyenne obtenue lors des 100 runs. La colonne `Best` y désigne la meilleure solution connue. Les graphes de convergence sont quant à eux repris aux FIGURES 7.5 et 7.6. Ceux-ci représentent l'évolution de la moyenne de la fonction objectif pour l'algorithme `Minamo SADistrEA`. La solution moyenne obtenue au moyen de l'algorithme `Minamo EA` est quant à elle représentée par une ligne horizontale. Notons qu'à la FIGURE 7.5b, nous ne traçons la courbe de convergence qu'à partir du cycle 1 car lors du cycle 0 (autrement dit, lors de la phase d'initialisation), le meilleur individu obtenu par chacun des 100 runs est non admissible. Il n'est donc pas cohérent de représenter la valeur de la fonction objectif de celui-ci. Pour tous les autres cas, les valeurs représentées correspondent à la valeur de la fonction objectif d'individus admissibles.

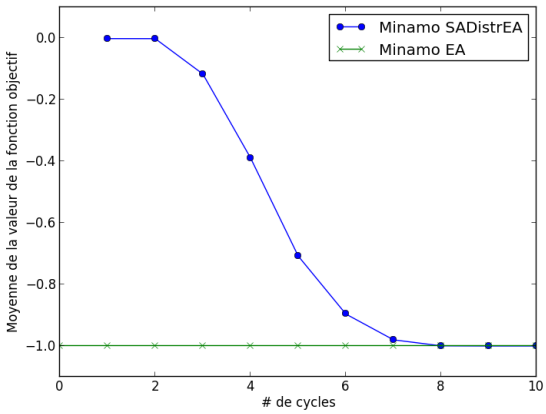
	`Best`	`EA`	`SADistrEA`
G2 - 20D	-8.03619e-01	-5.0027043e-01	**-7.3666485e-01**
G3MOD - 20D	-9.937e-01	**-9.9999962e-01**	-9.9998760e-01
G19 - 15D	3.26559e+01	1.1642325e+02	**3.6230885e+01**
RosenbrockMOD - 50D	1.2229690e+01	1.9716408e+02	**7.2935041e+01**
RosenbrockMOD - 500D	1.2474102e+02	6.6303874e+03	**8.0164117e+02**
SphèreMOD - 50D	5.0000000e+01	5.0000910e+01	**5.0000714e+01**
SphèreMOD - 500D	5.0000000e+02	5.0027370e+02	**5.0012155e+02**

TABLE 7.3 – Solutions moyennes des algorithmes `Minamo EA` et `Minamo SADistrEA`

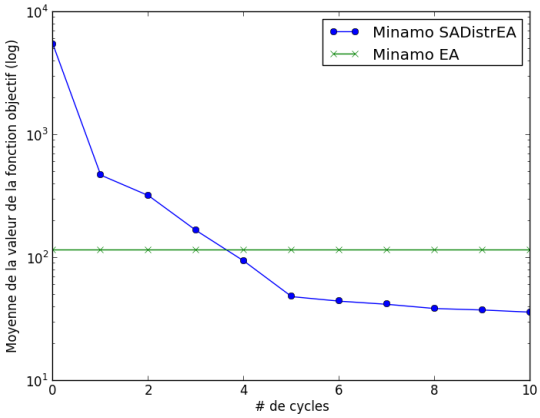
L'observation des solutions moyennes ainsi que des graphes de convergence nous mène au constat suivant : l'algorithme `Minamo SADistrEA` fournit également de meilleurs résultats que l'algorithme `Minamo EA` pour les problèmes contraints. En effet, les solutions moyennes obtenues par l'algorithme distribué sont meilleures (à l'exception du



(a) G2 - 20D

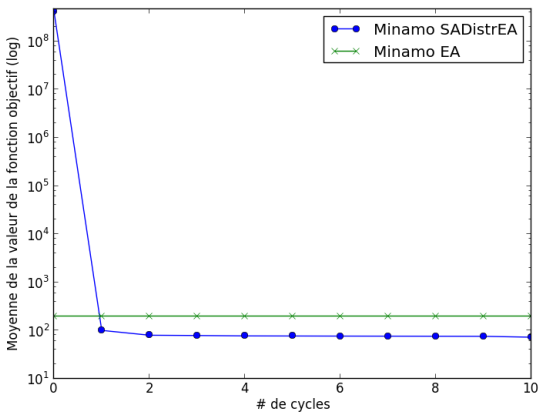


(b) G3MOD - 20D

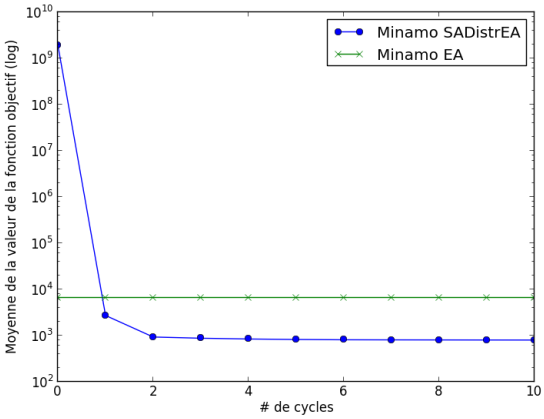


(c) G19 - 15D

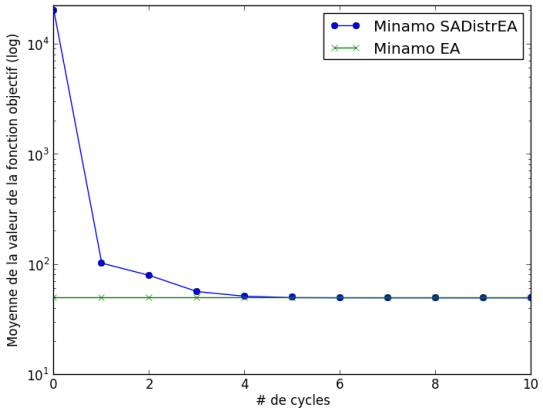
FIGURE 7.5 – Graphes de convergence des algorithmes `Minamo EA` et `Minamo SADistrEA`



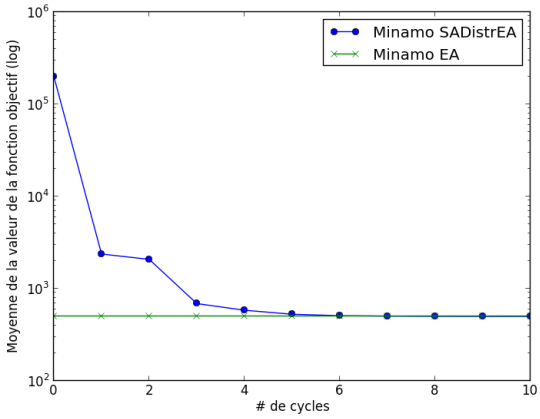
(a) RosenbrockMOD - 50D



(b) RosenbrockMOD - 500D



(c) SphèreMOD - 50D



(d) SphèreMOD - 500D

FIGURE 7.6 – Graphes de convergence des algorithmes `Minamo EA` et `Minamo SADistrEA`

problème G3MOD - 20D pour lequel les solutions sont relativement proches) que celles obtenues au moyen de l'algorithme classique. De plus, pour les problèmes G2 - 20D, RosenbrockMOD - 20 D et 50D, l'algorithme `Minamo SADistrEA` offre déjà de meilleurs résultats que l'algorithme `Minamo EA` après seulement un seul cycle. Il en est de même après seulement 4 cycles pour le problème G19 - 15D. Quant aux problèmes SphèreMOD - 50D et 500D, les deux algorithmes convergent vers la solution de ceux-ci, à savoir 50 pour SphèreMOD - 50D et 500 pour 500D.

Pour plus de détails sur ces résultats, les tableaux statistiques, reprenant les performances de chacun des algorithmes, sont fournis en Annexe aux Tables A4 et A5.

Conclusion et perspectives

Lors de ce mémoire, nous avons investigué, développé et testé une approche permettant d'optimiser des problèmes de grande dimension (nombre de variables allant de plusieurs dizaines à plusieurs centaines). Celle-ci est basée sur l'adaptation d'algorithmes évolutionnaires à l'optimisation de tels problèmes. Dans un premier temps, nous avons consulté la littérature afin d'établir un état de l'art des récentes recherches dans ce domaine. La plupart des publications référencent ce nouveau type d'algorithmes sous le nom d'*algorithme évolutionnaire distribué* ou *coopératif*. L'une d'entre elles, intitulée *Large scale evolutionary optimization using cooperative coevolution* et publiée par Yang, Tang et Yao [26], a particulièrement retenu notre attention. Ceux-ci y proposent un procédé itératif de décompositions aléatoires permettant de traiter de manière efficace la décomposition d'un problème de grande dimension en plusieurs sous-problèmes de dimensionnalité moindre.

Nous nous sommes ensuite appliqués à l'implémentation de ce procédé dans le logiciel MINAMO, plate-forme d'optimisation multi-disciplinaire du centre de recherche Cenaero. Deux algorithmes furent implémentés ; d'une part l'algorithme `Minamo DistrEA` qui reprend le procédé décrit par Yang, Tang et Yao et d'autre part, l'algorithme `Minamo SADistrEA`, développé dans le cadre de ce mémoire, permettant d'auto-adapter le nombre de sous-populations en cours d'optimisation. Ceux-ci font usage de nombreuses fonctionnalités implémentées dans le logiciel MINAMO, dont notamment l'algorithme évolutionnaire classique (i.e., non distribué) sur lequel repose le fonctionnement du logiciel.

Une fois ces algorithmes implémentés, nous avons testé la performance de ceux-ci sur des problèmes de grande dimension (50 et 500 D). Pour ce faire, nous avons commencé par construire un ensemble de fonctions test multi-dimensionnelles présentant des caractéristiques variées et couvrant ainsi un large éventail de problèmes d'optimisation mono-objectif sans contraintes. Nous avons ensuite appliqué nos algorithmes `Minamo DistrEA` et `Minamo SADistrEA` à l'ensemble des fonctions test et comparé leur performance à celle de l'algorithme évolutionnaire classique de MINAMO que nous avons nommé `Minamo EA`. Concernant l'algorithme `Minamo DistrEA`, l'interprétation des résultats fut on ne peut plus claire. Dans le contexte considéré, c'est-à-dire les problèmes de grande dimension, les résultats sont significativement meilleurs que ceux obtenus avec l'algorithme `Minamo EA`. Toutefois, le désavantage de cet algorithme réside dans le choix des paramètres (le nombre de sous-populations et le nombre de cycles) qui

doivent être choisis en adéquation avec la fonction objectif afin d'obtenir une optimisation de qualité. L'algorithme `Minamo SADistrEA` répond à cette problématique. Grâce à l'auto-adaptation du nombre de sous-populations en cours d'optimisation, celui-ci offre des résultats très satisfaisants et meilleurs que l'algorithme `Minamo DistrEA` pour la plupart des fonctions test et cela en fixant les paramètres identiques pour chacune d'entre elles. Nous avons également adapté ces algorithmes afin qu'ils puissent résoudre des problèmes d'optimisation avec contraintes. L'analyse des résultats obtenus pour de tels problèmes offre une perspective encourageante du potentiel de l'algorithme `Minamo SADistrEA` dans la résolution de problèmes contraints.

Finalement, ce mémoire ne constitue qu'une première approche du domaine d'application des algorithmes évolutionnaires distribués. D'une part, diverses améliorations pourraient être apportées aux algorithmes que nous avons développés. Nous pourrions notamment investiguer d'autres techniques de décomposition, entre autres basées sur les interactions entre variables ou encore considérer des décompositions avec *overlapping*, c'est-à-dire des décompositions permettant à une même variable de se retrouver dans différentes sous-populations. D'autre part, de nombreuses perspectives permettent d'élargir le domaine d'application des algorithmes évolutionnaires distribués. L'une d'entre elles répond à l'utilisation, de plus en plus fréquente, de simulations de haute-fidélité dans le monde de la recherche et de l'industrie. Une pratique courante et efficace afin de traiter des problèmes basés sur de telles simulations est de construire une succession de modèles de substitution (également appelés méta-modèles), de plus en plus précis, au sein d'une boucle d'optimisation. On parle alors d'optimisation assistée par méta-modèles. Rappelons que l'efficacité du logiciel Minamo repose principalement sur cette stratégie. L'une des perspectives est donc de développer, mettre en place et étudier une nouvelle méthode qui permette de résoudre, au moyen d'une optimisation distribuée/coopérative assistée par méta-modèles, des problèmes de grande dimension et basés sur des fonctions coûteuses. Il sera dès lors intéressant d'étudier les performances de l'algorithme distribué dans ce contexte. Une autre perspective est de répondre à la demande, issue de nombreuses applications du génie industriel, de traiter des problèmes pour lesquels on désire optimiser plusieurs objectifs tout en respectant certaines contraintes, en étudiant les algorithmes évolutionnaires multi-objectifs et y incluant l'aspect distribué/coopératif. Finalement, en addition au partitionnement opéré sur l'espace des entrées (variables/paramètres) considéré dans ce travail, il est également possible d'envisager un partitionnement de l'espace des sorties, à savoir les objectifs et les contraintes, afin de pouvoir résoudre des problèmes dits *many-objective*, comportant alors un grand nombre de sorties.

Bibliographie

[1] Affenzeller, M., Beham, A., Wagner, S., and Winkler, S. *Genetic Algorithms and Genetic Programming Modern Concepts and Pratical Applications.* CRC Press, 2009.

[2] Bellman, R. *Adaptive control Processes.* Princeton University Press, 1961.

[3] Bierlaire, M. *Introduction à l'optimisation différentiable.* Presses polytechniques et universitaires romandes, Lausanne, 2006.

[4] Cenaero. Simulation technologies for aeronautics, `http://www.cenaero.be` [En ligne ; Page disponible le 30-mars-2014].

[5] Chandra, R. *Problem Decomposition and Adaptation in Cooperative Neuroevolution.* PhD thesis, Victoria University of Wellington, 2012.

[6] Chen, W. *Large-Scale Numerical Optimization using Cooperative Coevolutionary Algorithms.* PhD thesis, University of Science and Technology of China, 2010.

[7] Coello, C. A. C. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms : a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering Vol. 191* (2002), pp. 1245–1287.

[8] Crélot, A.-S. Étude du couplage entre un algorithme génétique et des méthodes d'optimisation locale. Master's thesis, FUNDP, Namur, 2011.

[9] Deb, K. An efficient constraint handling method for genetic algorithms. In *Computer Methods in Applied Mechanics and Engineering* (1998), pp. 311–338.

[10] Hansen, N., Finck, S., Ros, R., and Auger, A. Real-parameter black-box optimization benchmarking 2009 : Noiseless functions definitions. Tech. Rep. RR-6829, INRIA, 2010.

[11] Hendrickx, R. Les algorithmes génétiques : Théorie et applications. Master's thesis, FUNDP, Namur, 2011.

[12] Holland, J. H. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[13] Leclercq, J. P. Optimisation combinatoire et discrète. Cours, FUNDP, Namur, 2010.

[14] Li, X., Tang, K., Omidvar, M. N., Yang, Z., and Qin, K. Benchmark functions for the CEC'2013 special session and competition on large scale global

optimization. Tech. rep., Evolutionary Computation and Machine Learning Group, RMIT University, Australia, 2013.

[15] Marelli, V. Matching on school choice : Theory and algorithms. Master's thesis, UNamur, 2013.

[16] Michalewicz, Z., Dasgupta, D., Le Riche, R. G., and Schoenauer, M. Evolutionary algorithms for constrained engineering problems. *Computers and Industrial Engineering Journal Vol. 30*, No. 2 (1996), pp. 851–870.

[17] NEOS. Optimization taxonomy, `http://www.neos-guide.org/content/optimization-taxonomy` [En ligne ; Page disponible le 30-avril-2015].

[18] NewcastleUniversity. Engineering design center, `http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/` [En ligne ; Page disponible le 21-avril-2014].

[19] Nocedal, J., and Wright, S. J. *Numerical Optimization, second edition*. Springer, USA, 2006.

[20] Omidvar, M. N., Li, X., Mei, Y., and Yao, X. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on evolutionary computation Vol. 10*, No. 10 (2013), pp. 1–17.

[21] Omidvar, M. N., Li, X., Yang, Z., and Yao, X. Cooperative co-evolution for large scale optimization through more frequent random grouping. *2010 IEEE Congress on Evolutionary Computation (CEC)* (2010), pp. 1–8.

[22] Potter, M. A., and De Jong, K. A. Cooperative coevolution : An architecture for evolving coadapted subcomponents. *Evolutionary Computation Vol. 8*, No. 1 (2000), pp. 1–29.

[23] Sainvitu, C., Iliopoulou, V., and Lepot, I. Global optimization with expensive functions - sample turbomachinery design application. In *Recent Advances in Optimization and its Applications in Engineering*, Springer, Ed. 2010.

[24] Saka, Y., Gunzburger, M., and Burkardt, J. Latinized, improved LHS, and CVT point sets in hypercubes. *International Journal of Numerical Analysis and Modeling Vol. 4*, No. 3-4 (2007), pp. 729–743.

[25] Tan, K. C., Yang, Y. J., and Goh, C. K. A distributed cooperative coevolutionary algorithm for multiobjective optimization. *IEEE Transactions on evolutionary computation Vol. 10*, No. 5 (2006), pp. 527–549.

[26] Yang, Z., Tang, K., and Yao, X. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, No. 178 (2008), pp. 2985–2999.

[27] Yao, X., Liu, Y., and Lin, G. Evolutionary programming made faster. *IEEE Transactions on evolutionary computation Vol. 3*, No. 2 (1999), pp. 82–102.

Annexe

	Meilleur	Pire	Moyenne	Médiane	Écart-type
Ackley - 50D	6.7441923e+00	1.1161667e+01	8.2892174e+00	8.4850756e+00	1.0696391e+00
Elliptique - 50D	5.2905793e-04	3.4837508e+05	6.7459396e+04	2.3301401e+04	9.2280295e+04
GenPen1 - 50D	6.2208468e-02	1.1469065e+02	3.7802000e+01	1.9202904e+01	3.7350295e+01
GenPen2 - 50D	2.2467325e-01	2.1768971e+01	8.9643024e+00	4.7982035e+00	7.3980474e+00
Griewank - 50D	3.6899439e-02	1.2081411e+00	3.4490642e-01	9.2274394e-02	3.9390365e-01
Quartique - 50D	3.3943769e-02	5.1480319e-02	4.1510990e-02	3.9529923e-02	5.9011324e-03
Rastrigin - 50D	4.4773380e+01	4.9749339e+01	4.8805579e+01	4.9747953e+01	1.6054694e+00
Rosenbrock - 50D	8.3222450e+01	2.3358887e+02	1.2482982e+02	1.1900930e+02	3.9806171e+01
Schwefel 12 - 50D	2.7005448e+01	2.7723721e+02	1.1500687e+02	9.3176540e+01	8.5025226e+01
Schwefel 2.21 - 50D	2.0000000e+01	2.0000000e+01	2.0000000e+01	2.0000000e+01	0.0000000e+00
Schwefel 2.22 - 50D	6.2773674e-07	1.1557880e+00	7.6739473e-02	1.7481936e-05	2.3912311e-01
Schwefel 2.26 - 50D	-1.6304160e+04	-9.9449895e+03	-1.3674101e+04	-1.3633218e+04	1.5499733e+03
Sphère - 50D	1.5461097e-05	3.2031386e-02	2.0962075e-03	3.6125945e-05	7.8140599e-03
Step - 50D	0.0000000e+00	2.3000000e+01	7.7157895e+00	2.0000000e+00	9.1879467e+00
AckleyTr - 50D	2.0039439e-05	1.9941608e+01	7.4604799e+00	4.1127296e+00	6.6004746e+00
ElliptiqueTr - 50D	6.4206333e+04	2.6832322e+07	4.6097146e+06	2.8054847e+06	5.7299056e+06
RastriginTr - 50D	4.3778199e+01	4.9751795e+01	4.9130236e+01	4.9747953e+01	1.2996291e+00
RosenbrockTr - 50D	4.6538053e+01	3.1646445e+03	4.8627212e+02	1.5504077e+02	8.7605640e+02
SphèreTr - 50D	3.1642599e-07	1.0371395e-02	1.4289255e-03	1.5777434e-04	2.8292768e-03
AckleyRot - 50D	9.1241087e+00	1.1732807e+01	1.0530490e+01	1.0823151e+01	7.7766086e-01
ElliptiqueRot - 50D	3.9798646e+05	1.6813733e+06	7.5605606e+05	7.8655494e+05	2.0773821e+05
RastriginRot - 50D	1.4924438e+01	4.8753095e+01	3.4530503e+01	3.3828593e+01	8.7439832e+00
Ackley - 500D	7.7799988e+00	9.5406036e+00	8.3549836e+00	8.2915007e+00	5.0258519e-01
Elliptique - 500D	2.7207867e+05	3.5380597e+06	9.0304496e+05	6.3668529e+05	6.2626513e+05
GenPen1 - 500D	7.9191047e-01	2.2196238e+01	7.8872095e+00	5.2299468e+00	6.8046963e+00
GenPen2 - 500D	1.9077944e+02	2.3159959e+03	4.6067287e+02	3.4867417e+02	3.5826193e+02
Griewank - 500D	1.6866196e+00	4.5009465e+00	2.3397531e+00	2.0016019e+00	9.4957080e-01
Quartique - 500D	3.6417276e-01	1.8438604e+00	7.4560966e-01	6.0140607e-01	4.0529514e-01
Rastrigin - 500D	1.2446395e+02	4.9749037e+02	4.1168691e+02	4.9747972e+02	1.5342519e+02
Rosenbrock - 500D	1.2092067e+04	1.4923519e+05	4.2719612e+04	3.0063319e+04	4.7151249e+04
Schwefel 12 - 500D	8.3970158e+03	1.9407206e+04	1.4746046e+04	1.5284896e+04	3.1571539e+03
Schwefel 2.21 - 500D	1.8193823e+01	2.0000000e+01	1.9695255e+01	2.0000000e+01	6.7928314e-01
Schwefel 2.22 - 500D	1.1313337e+00	4.9215199e+00	3.3710664e+00	3.8299872e+00	1.2153852e+00
Schwefel 2.26 - 500D	-1.2046014e+05	-2.7423884e+04	-8.2877394e+04	-1.0202186e+05	2.8188178e+04
Sphère - 500D	2.6830587e+01	4.2977188e+02	2.0209842e+02	2.1382046e+02	1.3290278e+02
Step - 500D	3.1380000e+03	8.8400000e+03	5.5927579e+03	5.8630000e+03	1.4832442e+03
AckleyTr - 500D	4.5578281e-06	1.0037876e+01	2.2437051e+00	3.4917033e-02	3.6576103e+00
ElliptiqueTr - 500D	1.9916772e+06	2.5103351e+07	8.1945486e+06	7.0463328e+06	4.8535965e+06
RastriginTr - 500D	1.2477406e+02	4.9748435e+02	2.5456579e+02	1.8221363e+02	1.4372621e+02
RosenbrockTr - 500D	9.9818217e+03	1.9552002e+05	3.6494346e+04	2.8051911e+04	3.2994098e+04
SphèreTr - 500D	4.4175794e+01	1.2082339e+03	3.3569831e+02	2.0903571e+02	3.2505713e+02
AckleyRot - 500D	7.6880402e+00	9.0652489e+00	8.2996023e+00	8.2875036e+00	3.8884124e-01
ElliptiqueRot - 500D	8.8770837e+06	1.5207429e+07	1.1848275e+07	1.1622194e+07	1.6959000e+06
RastriginRot - 500D	1.5685322e+02	2.6920734e+02	2.2569570e+02	2.3072991e+02	3.1989143e+01

TABLE A1 – Tableau statistique de l'algorithme `Minamo EA`

	Meilleur	Pire	Moyenne	Médiane	Écart-type
Ackley - 50D	1.8341639e-10	1.4288166e-09	3.8659919e-10	2.6036107e-10	3.5492288e-10
Elliptique - 50D	1.6497365e-27	2.1840517e-21	1.0685601e-22	1.2722188e-24	4.3892720e-22
GenPen1 - 50D	2.5260769e-29	1.6906163e-25	2.3746101e-26	5.3768801e-28	5.4547364e-26
GenPen2 - 50D	4.3514377e-29	6.1481201e-22	1.6373381e-23	1.9899908e-27	8.9366351e-23
Griewank - 50D	0.0000000e+00	3.6857488e-02	1.2874647e-02	9.8646721e-03	1.1103380e-02
Quartique - 50D	2.9171954e-02	4.8220916e-02	3.7719892e-02	3.2675530e-02	6.9418285e-03
Rastrigin - 50D	1.7053026e-13	3.9798366e+00	1.4348359e+00	9.9496001e-01	1.0735262e+00
Rosenbrock - 50D	7.1707221e-02	1.3568752e+02	4.4657126e+01	1.4256999e+01	4.6713959e+01
Schwefel 12 - 50D	3.0084156e-04	4.2858349e-03	1.1284238e-03	1.2005365e-03	5.9204877e-04
Schwefel 2.21 - 50D	5.9545015e-08	1.3610227e-07	8.5391820e-08	8.1715484e-08	2.1412699e-08
Schwefel 2.22 - 50D	1.6799204e-24	2.1555351e-20	2.4876855e-21	3.1537732e-22	5.2538796e-21
Schwefel 2.26 - 50D	-1.9883199e+04	-1.8577342e+04	-1.9404062e+04	-1.9283934e+04	3.9369881e+02
Sphère - 50D	2.7747635e-30	3.4381237e-27	6.9512707e-28	1.6228647e-28	1.1522244e-27
Step - 50D	0.0000000e+00	0.0000000e+00	0.0000000e+00	0.0000000e+00	0.0000000e+00
AckleyTr - 50D	5.0221916e-10	9.3280805e-10	6.8938606e-10	7.1537487e-10	9.9500778e-11
ElliptiqueTr - 50D	2.1580358e-14	9.9661616e-14	5.6578530e-14	5.8261602e-14	2.1983782e-14
RastriginTr - 50D	0.0000000e+00	4.9747953e+00	2.2070429e+00	1.9899181e+00	1.7409746e+00
RosenbrockTr - 50D	1.4606468e+01	1.0048053e+02	4.4709111e+01	4.4591619e+01	1.9811512e+01
SphèreTr - 50D	9.5142086e-30	1.6423065e-26	1.4112576e-27	7.1052466e-28	2.7238326e-27
AckleyRot - 50D	1.5612830e+00	3.2224559e+00	2.2013805e+00	2.0781074e+00	4.8467306e-01
ElliptiqueRot - 50D	1.1714499e+05	5.0567203e+05	2.6553965e+05	2.2834423e+05	1.0441039e+05
RastriginRot - 50D	2.9848762e+01	7.4621839e+01	4.0322002e+01	3.8803363e+01	8.5291723e+00
Ackley - 500D	3.6380446e-03	3.9979627e-03	3.8251430e-03	3.8460944e-03	1.1567143e-04
Elliptique - 500D	2.0338966e-06	4.4762476e-05	1.1389644e-05	1.0332182e-05	8.1994539e-06
GenPen1 - 500D	5.4844831e-15	4.8634277e-08	3.0736311e-09	7.5686305e-15	1.1892374e-08
GenPen2 - 500D	1.1143834e-12	4.1080968e-12	1.7608379e-12	1.4786295e-12	8.0216548e-13
Griewank - 500D	5.6709082e-12	9.8646721e-03	1.1422252e-03	6.4752648e-12	3.1731668e-03
Quartique - 500D	2.5712212e-01	3.4333285e-01	2.9938697e-01	3.0127003e-01	2.6177144e-02
Rastrigin - 500D	8.7749186e-09	4.9747953e+00	1.6927300e+00	1.3346543e+00	1.1502641e+00
Rosenbrock - 500D	5.4367288e+02	8.5251434e+02	7.1871933e+02	7.0904119e+02	8.3918805e+01
Schwefel 12 - 500D	6.1198623e+02	1.0900473e+03	8.7209365e+02	8.7369218e+02	1.3187031e+02
Schwefel 2.21 - 500D	3.5660049e-03	5.5605933e-03	4.7193498e-03	5.0722816e-03	7.3132561e-04
Schwefel 2.22 - 500D	7.5051025e-11	4.6696713e-09	1.1583291e-09	6.5268297e-10	1.3273123e-09
Schwefel 2.26 - 500D	-1.9481509e+05	-1.8993998e+05	-1.9225367e+05	-1.9235186e+05	1.1949966e+03
Sphère - 500D	2.0553893e-19	6.2991146e-19	3.9208333e-19	3.5414503e-19	1.2999720e-19
Step - 500D	0.0000000e+00	0.0000000e+00	0.0000000e+00	0.0000000e+00	0.0000000e+00
AckleyTr - 500D	3.5277919e-03	3.8995895e-03	3.7213526e-03	3.6993493e-03	1.2269647e-04
ElliptiqueTr - 500D	1.1971648e+01	1.6045809e+01	1.4601956e+01	1.4734077e+01	1.1223406e+00
RastriginTr - 500D	1.2258580e-01	1.1782998e+00	7.0045334e-01	1.1508853e+00	4.8536400e-01
RosenbrockTr - 500D	4.7140581e+02	5.9345104e+02	5.1927024e+02	5.2047537e+02	3.1453650e+01
SphèreTr - 500D	3.2979261e-19	1.1032689e-18	6.1304737e-19	5.9795952e-19	1.7207275e-19
AckleyRot - 500D	2.8532073e-03	3.2268597e-03	3.0771901e-03	3.0568870e-03	9.0423788e-05
ElliptiqueRot - 500D	1.3656624e+06	2.2670717e+06	1.8099522e+06	1.7928444e+06	2.4879289e+05
RastriginRot - 500D	2.4078003e+02	5.3727736e+02	3.9389881e+02	4.0594306e+02	8.4653837e+01

TABLE A2 – Tableau statistique de l'algorithme `Minamo DistrEA`

	Meilleur	#sp	Pire	#sp	Moyenne	Médiane	Écart-type
Ackley - 50D	3.9968029e-15	2	7.5495166e-15	2	7.3625316e-15	7.5495166e-15	7.9751844e-16
Elliptique - 50D	2.9892063e-42	10	6.0952156e-37	10	1.2708699e-38	4.3035039e-40	6.7027538e-38
GenPen1 - 50D	9.4232686e-33	2	9.4232686e-33	2	9.4232686e-33	9.4232686e-33	2.7514306e-48
GenPen2 - 50D	1.3497838e-32	2	1.3497838e-32	2	1.3497838e-32	1.3497838e-32	1.9260014e-47
Griewank - 50D	0.0000000e+00	25	3.1971690e-02	2	3.0590474e-03	0.0000000e+00	6.9007618e-03
Quartique - 50D	2.9836903e-02	2	6.2582984e-02	2	4.5464758e-02	4.5712903e-02	8.9032999e-03
Rastrigin - 50D	0.0000000e+00	25	1.9899181e+00	2	4.6082314e-01	0.0000000e+00	5.9504106e-01
Rosenbrock - 50D	2.4411169e-01	2	1.4204826e+02	2	3.7861053e+01	1.2795137e+01	4.0741337e+01
Schwefel 12 - 50D	2.1596867e-01	2	5.5583247e+00	2	2.3285494e+00	1.8587873e+00	1.4732979e+00
Schwefel 2.21 - 50D	1.8746772e-10	25	7.0697023e-10	2	3.8736625e-10	3.5507264e-10	1.6097045e-10
Schwefel 2.22 - 50D	1.3646230e-26	10	1.1974749e-24	10	2.7886295e-25	1.2327343e-25	3.4137723e-25
Schwefel 2.26 - 50D	-2.0593829e+04	2	-1.9883199e+04	2	-2.0262042e+04	-2.0238514e+04	2.1416507e+02
Sphère - 50D	2.8459471e-46	10	3.7015882e-40	10	2.6889505e-41	5.8114390e-44	8.6548853e-41
Step - 50D	0.0000000e+00	25	0.0000000e+00	25	0.0000000e+00	0.0000000e+00	0.0000000e+00
AckleyTr - 50D	3.9968029e-15	2	7.5495166e-15	2	7.0633557e-15	7.5495166e-15	1.2274762e-15
ElliptiqueTr - 50D	1.4075481e-42	10	1.7696719e-35	10	1.2977904e-36	1.5357643e-39	3.9728773e-36
RastriginTr - 50D	0.0000000e+00	25	1.9899181e+00	2	4.8177011e-01	0.0000000e+00	6.3032386e-01
RosenbrockTr - 50D	2.3880741e+01	2	1.0542804e+02	2	6.4247288e+01	7.6670969e+01	2.7818490e+01
SphèreTr - 50D	2.9940595e-45	10	1.2419542e-39	10	9.8492062e-41	8.5334402e-43	3.0013887e-40
AckleyRot - 50D	3.9968029e-15	2	7.5495166e-15	2	7.0259588e-15	7.5495166e-15	1.2660213e-15
ElliptiqueRot - 50D	6.1958152e+05	2	1.7801019e+08	2	3.8072797e+07	1.9872349e+07	4.5751754e+07
RastriginRot - 50D	5.1737841e+01	2	1.8307190e+02	2	9.9859252e+01	9.6510877e+01	3.0157195e+01
Ackley - 500D	2.1715962e-13	50	3.0597747e-13	50	2.6005397e-13	2.5623947e-13	1.8396630e-14
Elliptique - 500D	5.5244966e-20	50	1.2313825e-15	50	1.1647897e-16	1.6354215e-17	3.1282385e-16
GenPen1 - 500D	8.7512924e-27	50	9.0727846e-25	50	3.8904831e-25	4.2728370e-25	2.6521220e-25
GenPen2 - 500D	1.7748933e-25	50	4.3778549e-22	50	1.0732272e-22	1.0884599e-22	1.1192849e-22
Griewank - 500D	1.1102230e-16	25	2.2204460e-16	25	2.0918939e-16	2.2204460e-16	3.5712519e-17
Quartique - 500D	2.6502282e-01	2	3.2375803e-01	2	2.9350369e-01	2.9523084e-01	1.3704448e-02
Rastrigin - 500D	0.0000000e+00	100	2.2737368e-13	100	1.3163739e-14	0.0000000e+00	4.0153837e-14
Rosenbrock - 500D	3.6137042e+00	2	1.8510953e+01	2	1.0324862e+01	9.0855515e+00	4.3473605e+00
Schwefel 12 - 500D	1.0928210e+04	2	2.6641059e+04	2	1.5553263e+04	1.4703603e+04	3.6363829e+03
Schwefel 2.21 - 500D	7.8847212e-02	2	1.1177007e+00	2	3.0406840e-01	3.1993463e-01	2.2415627e-01
Schwefel 2.22 - 500D	9.6375196e-13	50	1.6216881e-12	50	1.1356821e-12	1.1204799e-12	1.4956286e-13
Schwefel 2.26 - 500D	-2.0821641e+05	2	-2.0557620e+05	2	-2.0651095e+05	-2.0636898e+05	7.3151171e+02
Sphère - 500D	1.0161080e-22	50	7.9423576e-21	50	3.2558977e-21	3.9668754e-21	2.1922000e-21
Step - 500D	0.0000000e+00	250	0.0000000e+00	250	0.0000000e+00	0.0000000e+00	0.0000000e+00
AckleyTr - 500D	2.4558133e-13	50	3.5216274e-13	50	3.1177400e-13	3.1308289e-13	2.6454702e-14
ElliptiqueTr - 500D	7.5024998e-17	50	1.1139575e-14	50	3.4438116e-15	1.7670570e-16	3.8685452e-15
RastriginTr - 500D	0.0000000e+00	100	0.0000000e+00	100	0.0000000e+00	0.0000000e+00	0.0000000e+00
RosenbrockTr - 500D	3.6264772e+02	2	8.9552782e+02	2	6.2843666e+02	6.4746759e+02	1.3982467e+02
SphèreTr - 500D	3.0007794e-22	50	2.1023601e-20	50	1.2792947e-20	1.4043682e-20	5.7049016e-21
AckleyRot - 500D	2.6689762e-13	50	1.0023093e-12	50	6.0964099e-13	7.1809225e-13	2.4104972e-13
ElliptiqueRot - 500D	4.0059997e+06	2	6.3156586e+06	2	5.2465463e+06	5.0299721e+06	7.1273054e+05
RastriginRot - 500D	5.1343069e+02	2	7.9510678e+02	2	6.8350484e+02	6.5671945e+02	9.7501191e+01

TABLE A3 – Tableau statistique de l'algorithme `Minamo SADistrEA`

	Meilleur	Pire	Moyenne	Médiane	Écart-type
G2 - 20D	-6.3172125e-01	-3.7052188e-01	-5.0027043e-01	-4.9728684e-01	6.2741303e-02
G3MOD - 20D	-9.9999991e-01	-9.9999885e-01	-9.9999962e-01	-9.9999967e-01	2.2045730e-07
G19 - 15D	3.2847046e+01	6.4588873e+02	1.1642325e+02	4.0203650e+01	1.7113612e+02
RosenbrockMOD - 50D	7.4760698e+01	5.3802380e+02	1.9716408e+02	1.8667887e+02	8.8880160e+01
RosenbrockMOD - 500D	3.6887687e+03	1.3555667e+04	6.6303874e+03	5.6683627e+03	2.6621190e+03
SphèreMOD - 50D	5.0000195e+01	5.0002263e+01	5.0000910e+01	5.0000918e+01	4.7214756e-04
SphèreMOD - 500D	5.0009142e+02	5.0128173e+02	5.0027370e+02	5.0018048e+02	2.2938776e-01

Table A4 – Tableau statistique de l'algorithme `Minamo EA`

	Meilleur	Pire	Moyenne	Médiane	Écart-type
G2 - 20D	-7.9489683e-01	-6.4928864e-01	-7.3666485e-01	-7.4193476e-01	3.9235365e-02
G3MOD - 20D	-9.9999830e-01	-9.9993855e-01	-9.9998760e-01	-9.9999450e-01	1.4567861e-05
G19 - 15D	3.3447839e+01	3.9696643e+01	3.6230885e+01	3.5224214e+01	1.8784618e+00
RosenbrockMOD - 50D	2.4719056e+01	1.4453957e+02	7.2935041e+01	8.4748535e+01	2.8015848e+01
RosenbrockMOD - 500D	5.6871546e+02	9.6628003e+02	8.0164117e+02	8.0859803e+02	1.2166860e+02
SphèreMOD - 50D	5.0000276e+01	5.0001755e+01	5.0000714e+01	5.0000707e+01	2.7412874e-04
SphèreMOD - 500D	5.0006999e+02	5.0029455e+02	5.0012155e+02	5.0009425e+02	6.1269657e-02

Table A5 – Tableau statistique de l'algorithme `Minamo SADistrEA`